34

NOTE DI SOFTWARE

UNIVERSITÀ
DEGLI STUDI DI MILANO
DIPARTIMENTO DI SCIENZE
DELL'INFORMAZIONE

Honeywell
Honeywell Information Systems Italia

Note di software.
è un bollettino trimestrale, edito a cura del Centro di Ricerca e Progettazione della Honeywell Information Systems Italia e del Dipartimento di Scienze dell'Informazione dell'Università di Milano.

Note di software.
intende costituire uno strumento per la rapida e informale circolazione di informazioni, idee ed esperienze nell'area della tecnologia del software. In particolare, intende stimolare il dibattito sulle metodologie orientate alla diminuzione del rapporto costo/qualità dei programmi, e sugli strumenti atti ad automatizzare l'attività di produzione del software in ogni suo aspetto.

La collaborazione a **Note di software** è libera.
In particolare, si sollecitano contributi nella forma di brevi monografie e bibliografie essenziali commentate sui seguenti temi: tecniche e strumenti per la programmazione strutturata e modulare, metodologie e strumenti per il testing e il debugging dei programmi, documentazione del software, aspetti organizzativi e gestionali nella costruzione di sistemi software di grandi dimensioni.

Chi desiderasse pubblicare un contributo è pregato di prendere contatto con il Comitato di Redazione (HISI - Via ai Laboratori Olivetti - Pregnana Milanese), o di inviare direttamente il dattiloscritto in triplice copia.
I contributi non dovrebbero superare le quattromila parole.

I lavori accettati per la pubblicazione vengono revisionati dal Comitato di Redazione, che si avvale del contributo di specialisti del settore. Ogni contributo pubblicato riflette, comunque, le opinioni dei suoi autori, e non necessariamente quelle del Comitato di Redazione. Eventuali revisioni saranno effettuate di comune accordo fra il Comitato di Redazione e i proponenti.

Note di software.
Viene distribuito ad aziende, Enti e specialisti del settore che ne facciano richiesta alla redazione.

Direttore responsabile: Paolo Ghelfi
Comitato di Redazione: Roberto Polillo - Wanda Anselmetti - Stefania Bandini
Redazione: Franco Soresini

ARCHIVIO STORICO
ASSOCIAZIONE POZZO DI MIELE
FDPM-86 RNSw123

34

FPDM-86

NOTIZIARIO SOFTWARE

Dicembre 1986

Indice:

QUESTO NUMERO Pag. 1

— EASYLIFE: UNA FACILE VIA D'ACCESSO AL MONDO UNIX, di N. Bosetti e L. Scura " 3

— SCIENZA DEL SOFTWARE: METODI E RISULTATI, di L. Felician e G. Zalateu " 13

— ELECTRONIC MAIL SYSTEMS AND COMMUNICATIVE BEHAVIOUR, di G. Tonfoni " 19

— UNA COMPONENTE LINGUISTICA PER UN SISTEMA DI QUESTION ANSWERING, di M.R. Savoldi " 24

— PROTOTIPIZZAZIONE ED EXPERT SYSTEM SHELL: UN ESEMPIO D'USO DI Xi, di S. Marioni " 30

AVVENIMENTI

— THE 2nd ADVANCED COURSE ON PETRI NETS, di F. De Cindio e L. Pomello " 37

IL NOTIZIARIO SUI CD-ROM " 40

RECENSIONI

— a cura di E. Ciapessoni " 41

Autorizzazione del tribunale di Milano n. 87 del 23 febbraio 1977

QUESTO NUMERO

Questo numero di NOTE DI SOFTWARE si apre con un articolo di Nicoletta Bosetti e Lucia Scura in cui viene proposto un sistema d'accesso all'ambiente UNIX, orientato all'utente, al fine di rendere più semplice ed agevole l'uso dello stesso. Questo tema è di particolare interesse, visto anche il crescente impiego di tale sistema operativo da parte di utenti non specialisti.

Il secondo contributo, di Leonardo Felician e Graziella Zalateu, tratta un argomento di crescente interesse: la "Scienza del Software", ossia una teoria che si occupa della misura di alcune grandezze relative a programmi mediante le quali è possibile fare valutazioni sulla "qualità" degli stessi. Gli esperimenti condotti su alcuni campioni scritti in Pascal hanno costituito interessanti verifiche sperimentali della teoria.

Graziella Tonfoni tratta invece nel suo articolo il problema estremamente attuale della definizione di modelli interpretativi delle dinamiche di comunicazione all'interno di un'organizzazione, quale ad esempio un ufficio. A prescindere dalle conclusioni più prettamente tecniche a cui giunge l'autrice, sono comunque fondamentali: l'osservazione che l'uso degli strumenti informatici di Office Automation deve avvenire "in campo" e la metodologia impiegata per condurre l'indagine.

Il tema proposto da Maria Rosa Savoldi, un sistema di dialogo uomo-macchina basato su un linguaggio pseudo-naturale, testimonia il crescente interesse per interfacce più vicine all'utente, anche tramite un linguaggio pseudo-naturale, e la naturalezza con la quale la trattazione automatica del linguaggio sia resa possibile dal Prolog.

L'ultimo articolo di questo numero, di Silvano Marioni, presenta un'esperienza sull'uso di uno "shell" per la costruzione di sistemi esperti: la realizzazione di un'applicazione nel settore bancario, in particolare la tariffazione delle commissioni di borsa. Le osservazioni dell'autore rappresentano un'interessante testimonianza di utente di "shell".

Tra gli AVVENIMENTI vi è il resoconto dettagliato da parte di Fiorella De Cindio e Lucia Pomello del "The 2nd Advanced Course on Petri Nets", che ha avuto luogo in Germania nel settembre dello scorso anno.

Il NOTIZIARIO SUI CD-ROM, curato da A. Borgia, G. Dalto, G. Ferrari, G. Pagani e F. Vagliasindi propone una serie di interessanti informazioni sul mondo CD-ROM.

Infine, rubrica dedicata alle RECENSIONI di NOTE DI SOFTWARE, a cura di E. Ciapessoni, presenta rispettivamente un commento sul testo "INTRODUZIONE ALL'ARCHITETTURA DEI SISTEMI DI ELABORAZIONE" di Pierluigi Fiorani Gallotta, e sul libro "DOCUMENTAZIONE AUTOMATICA" di Dario Lucarella.

La REDAZIONE

EASYLIFE (*): UNA FACILE VIA D'ACCESSO AL MONDO UNIX (**)

Lucia Scura, Nicoletta Bosetti

Honeywell Information Systems Italia

Pregnana (MI)

RIASSUNTO

L'interfaccia utente costituisce il punto di contatto tra l'utente e il sistema determinando il tipo di colloquio uomo-maccina. Unix ha un'interfaccia utente a comandi, la shell. Poichè tali comandi non sono di facile utilizzo risulta complesso per l'utente inesperto colloquiare con il sistema.
La nostra proposta è una "ease of use" per Unix, orientata all'utente inesperto, il cui scopo è di consentire di utilizzare il maggior numero di funzioni senza dover preliminarmente leggere diverse pagine di manuale.
Questa interfaccia permette un tipo di colloquio attraverso la selezione di voci da menu ed una sintassi che, a fronte di ogni azione, propone gli oggetti sui quali l'azione è ammessa.
Alcune caratteristiche di questa "ease of use" sono:

— la personalizzazione dei menu, in modo che ogni utente abbia le funzioni a cui è interessato;
— la gestione delle lingue nazionali, per cui ogni utente può utilizzare la lingua che preferisce per colloquiare col sistema;
— un metodo semplice e logico per trovare le funzioni richieste guidato da cammini gerarchici.

Queste, ed altre caratteristiche, sono unite ad una estrema facilità d'uso.

ABSTRACT

The user interface is the contact point between the user and the system. It determines the man-machine dialogue. Unix has a command language user interface called the shell. Because it is difficult to remenber the shell commands, it is not easy for the user to communicate with the system.
Our proposal is an "ease of use" interface for Unix, oriented toward the end-user.
The aim is to allow the maximum usage of system functionality, without any detailed kwonledge of the Unix commands.
In this interface, the system commands are organized in hierarchical menues, in five groupes: Services, File management, User Applications, Software Factory and System Administration.
The interface allows dialogue through the selection of menu options. Its syntax causes the related file for each function to be displayed.
Tha main features of this "ease of use" interface are:

— the ability to personalize the menues, so that the user has only those functions required, and not a system cluttered with unused and unwanted commands;
— the ability to use differnt national language, so that the user may work in a confortable enviroment;
— a simple and logical method to find the required functionality by guided hierarchical descent.

These, and other features, provide the user with a greatty enhanced ease of use.

1. INTRODUZIONE

Unix è un sistema operativo nato in ambiente universitario, per cui alcune sue caratteristiche sono:

— un'interfaccia verso lo user con una sintassi estremamente complessa;
— alta flessibilità e quindi assenza di controlli;
— diagnostica di errore molto scarsa;
— documentazione vasta e in formato dizionario.

La diffusione sempre maggiore di questo sistema ha comportato un cambiamento in quello che era l'utente abituale: dallo studente ricercatore esperto di informatica all'utente generico, inesperto, che usa il sistema operativo per

(*) Easylife is a Trademark H.I.S.I.
(**) UNIX is a Trademark of Bell Laboratories

facilitare il proprio lavoro, e che si aspetta quindi un prodotto affidabile, semplice da usare e tale da soddisfare le proprie esigenze.

Di qui la necessità di creare un'interfaccia che renda Unix "easy to use" anche per l'utente meno esperto.
L'esigenza è che Unix sia usato dalla segretaria e dal commercialista, negli uffici bancari e nelle industrie, nel commerciale e nel gestionale.
Una delle difficoltà maggiori sembre quella di ricordare il nome dei comandi e delle loro numerose opzioni. Infatti, mentre alcuni comandi sono abbastanza intuitivi altri sono invece molto complessi e non mnemonici.

Ad esempio per chiedere la Posta o la data corrente è sufficiente digitare "mail" e "date"; invece per memorizzare un file su un dischetto bisogna digitare "tar cvf / dev / xxx nomefile" e questo è un processo più complesso.

È necessaria quindi una certa dimestichezza col sistema prima di acquisirne facilità d'uso.
Reperire le informazioni necessarie per l'utilizzo del sistema è un altro fondamentale problema.
I manuali di sistema sono per lo più raccolte di comandi ordinati alfabeticamente. L'utente deve avere una precisa idea di quello che vuole cercare e deve conoscere il nome del comando che vuole eseguire.
Leggere un manuale Unix e pretendere di saper usare il sistema è praticamente impossibile almeno per un utente che utilizza Unix come supporto ad una attività non informatica. Anche chi, pur lavorando in ambiente software, utilizza Unix solo occasionalmente, trova delle difficoltà.

Oltretutto un utente vede malvolentieri un sistema che presenta volumi di manuali da leggere ed è sicuramente più attratto da un sistema in cui le possibilità di movimento e il tipo di operazioni applicabili sono descritte, in ogni situazione, direttamente dall'interfaccia stessa.
Anche la lingua utilizzata ha il suo peso. La maggior parte dei manuali è scritta in lingua inglese e questo contribuisce a creare difficoltà negli utenti non anglosassoni.

2. PROPOSTA DI UN'INTERFACCIA UTENTE PER UNIX

Il nostro scopo è quello di proporre un'interfaccia utente per il sistema operativo Unix che offra il maggior numero di funzioni di sistema organizzate in sezioni (menu) che ne identificano classi omogenee.
Il progetto sviluppato si chiama EASYLIFE, parola composta dalla iniziali dei seguenti termini:

EASY
(multi) Language
Interface
For
End user

è un'interfaccia uomo-macchina che si pone come interprete del colloquio tra il sistema operativo Unix e l'utente attraverso un'organizzazione a finestre e menu, proponendosi di utilizzare un liguaggio il più vicino possibile alla lingua dell'utente.
Fondamentalmente Easylife è una shell che permette il colloquio con l'uente in maniera immediata, organizzando i comandi Unix in modo che egli possa utilizzare il sistema senza dover prima acquisire un livello di "know how" approfondito.

3. IL MODELLO

Nella difinizione di interfaccia utente un aspetto fondamentale è riuscire a cogliere il modello a cui l'utente si riferisce quando deve eseguire una certa operazione col sistema.
Conseguentemente bisogna costruire un'interfaccia che sia il più vicino possibile alla rappresentazione della macchina come è vista dall'utente.

Per costruire un'interfaccia che sia compatibile con il modello mentale dell'utente bisogna definire un modello concettuale del sistema.
Volendo formulare un modello concettuale del sistema operativo Unix potremme considerare questo come la combinazione di due fattori: oggetti e azioni.

Le azioni sono tutti quei file che possono essere eseguiti e corrispondono, nella terminologia Unix ai comandi.

Gli oggetti sono tutti gli altri file.

In Unix sia gli oggetti sia le azioni non sono presentati secondo alcun ordine funzionale per l'end-user in quanto seguono la struttura ad albero del file system.

4. IL MODELLO DI EASYLIFE

Easylife propone un modello del sistema nel quale le azioni di Unix sono raggruppate per funzionalità e gli oggetti per tipi.

4.1 Le azioni

In Easylife le azioni sono ordinate in modo che l'utente sia guidato nel percorso necessario per raggiungere un determinato scopo.
Sono state suddivise per funzionalità in cinque gruppi all'interno dei quali l'azione perde la corrispondenza con il nome del programma che la esegue, e viene sempre riferita con la funzione che rappresenta. Si distinguono:

Servizi
Gestione File
Applicazioni
Produzione di Software
Amministrazione del Sistema

Servizi

Sono un insieme di funzioni le quali forniscono all'utente

alcuni strumenti di uso immediato (calcolatrice, posta elettronica, calandario, gestione degli errori ...).

Gestione file

Consiste di tutte le funzioni che sono applicabili agli oggetti visti dal file system di Unix (cancella, crea, copia, confronta, rimuovi ...).

Applicazioni

Sono sia le applicazioni fornite direttamente dal sistema (nroff,uucp,cu), sia quelle introdotte come pacchetti (word processor, pacchetti grafici, data-base, pacchetti di comunicazione).

Produzione di software

È una sezione indirizzata ai programmatori e contiene i compilatori di cinque linguaggi (C-language, Pascal, COBOL, FORTRAN, BASIC) e numerosi tool disponibili per la software factory.

Amministrazione di sistema

Questa sezione, indirizzata al system administrator, comprende operazioni quali creazione di un utente, configurazione di terminali dei processi attivi, configurabilità del disco di sistema ...

4.2 Gli oggetti

Gli oggetti in Easylife sono organizzati in tipi che rispecchiano, per quanto possibile la tipologia riconosciuta in Unix; cioè anche in Easylife esistono le direttrici, i file eseguibili, i file leggibili, i file rimuovibili, i file copiabili, i file modificabili, ecc... e per quanto sia possibile si distinguono i file nei vari linguaggi di programmazione.
Vengono proposti ogni volta solo gli oggetti relativi ad una determinata azione. Ad esempio, se l'azione è "esegui", verranno proposti solo i file eseguibili, mentre se l'azione è "visualizza" verranno proposti i file leggibili, ... in modo che non si rischi di voler eseguire file non eseguibili oppure di editare file non editabili. In questo modo l'utente ha meno possibilità di fare errori.

5. SINTASSI

Uno dei principali scopi della fase "ease of use" proposta è quello di fornire una sintassi che metta in relazione le azioni e gli oggetti.
Si sono analizzate tre possibili rappresentazioni:

— un insieme di oggetti e, a fronte di ogni selezione, le azioni relative ad ogni oggetto;
— un insieme di azioni e, a fronte di ognuna di esse, gli oggetti ai quali le azioni sono applicabili;
— un insieme di azioni e di oggetti contemporaneamente.

Decidere quale possibilità scegliere non è banale, perchè non bisogna dimenticare l'obiettivo del progetto: fornire un modello che sia vicino al modello mentale dell'utente. L'utente quando decide di lavorare ha nella mente un'azione da compiere su un oggetto, oppure un oggetto a cui associare un'azione? In realtà l'azione o l'oggetto nella mente umana vengono creati quasi contemporaneamente. È quindi difficile forzare l'utente a pensare prima all'oggetto e poi all'azione o viceversa, ed è altrettanto difficile presentarli insieme.

La nostra scelta è caduta comunque sulla sequenza azione-oggetto perchè ritenuta di più immediata comprensione. A fronte di ogni azione vengono proposti gli oggetti su cui l'azione è ammessa. Questo riduce drasticamente la possibilità di errore da parte dell'utente e contribuisce ad offrire un'immagine di qualità del sistema offrendo tutta una serie di cammini prestabiliti e sicuramente funzionanti.

Facciamo un esempio (fig. 1).

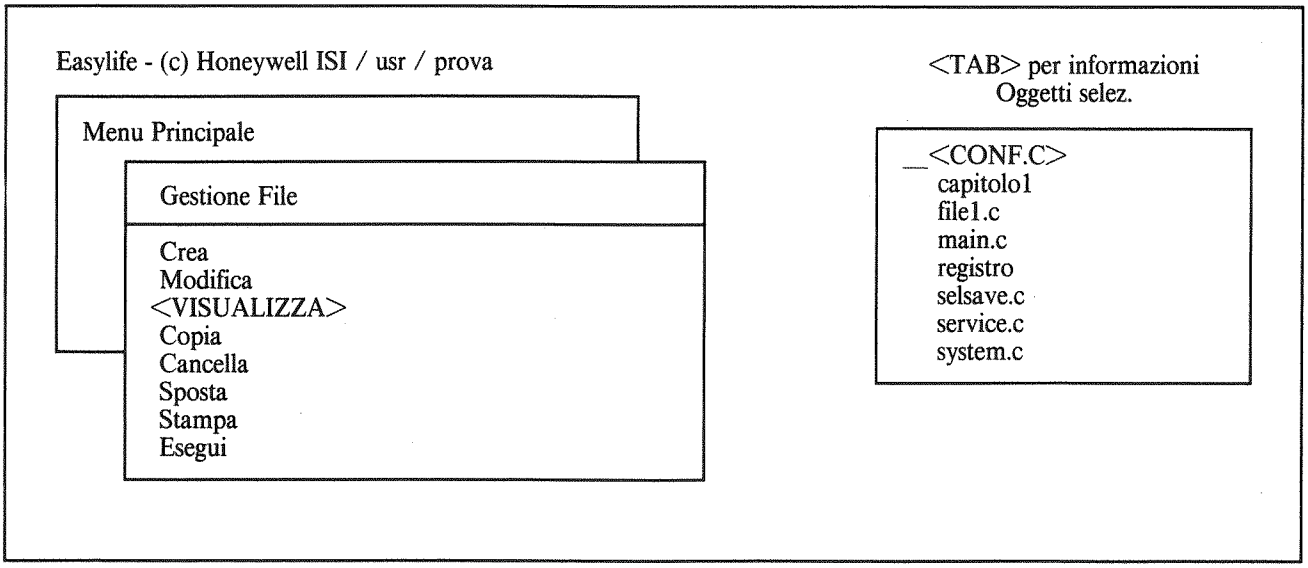


Fig. 1 Esempio di finestre sul terminale.

Selezionando la funzione "gestione file" appare il menu relativo a questa funzione dove sono contenute le azioni che si possono compiere sui file. Selezionando ora l'azione "visualizza" vengono proposti tutti gli oggetti della direttrice corrente visualizzabili. La scelta degli oggetti segue la tipizzazione propria di Unix, per cui ci si riferisce alle protezioni per sapere se i file sono visualizzabili (fig. 1).

6. I COMANDI

La maggior parte delle azioni di Easylife viene realizzata attraverso l'uso di comandi Unix. Ossia viene mandato in esecuzione il comando con cui i suoi argomenti. Alcuni comandi necessitano di particolari opzioni ch evengono aggiunte direttamente da Easylife per facilitare l'utente inesperto.

Alcuni comandi poi vengono eseguiti in **background,** perchè altrimenti bloccherebbero l'utente per un tempo troppo lungo: è il caso del comando "nroff". In altri viene chiesto, tramite un menu, quale tipo di esecuzione si desidera.

Per i comandi che hanno un'uscita su "standard output" esiste un menu che permette la ridirezione dell'output su stampante, video o file.
In alcuni casi, comunque, è stato ,ecessario implementare nuovi comandi o perchè inesistenti in Unix o perchè quelli esistenti non erano adatti ad essere inseriti in Easylife.
Un esempio a questo proposito è dato dal comando di **Backup** che gestisce il salvataggio selettivo di file su più dischetti e il ripristino successivo su disco. Un altro esempio è l'implementazione del comando per l'**invio dei messaggi** tra gli utenti. Questo comando (msgsnd) in Unix è gestito in modo che ogni linea terminata con un carattere di return venga accodata per essere inviata. In questo modo non è possibile correggere errori che non siano sulla stessa linea.

Col nuovo comando il messaggio viene scritto interamente in una specifica finestra prima di essere inviato. È possibile così correggere errori e mandare messaggi su più linee, è possibile inoltre inviare il messaggio ad uno specifico terminale oltre che ad uno specifico utente (fig. 2).

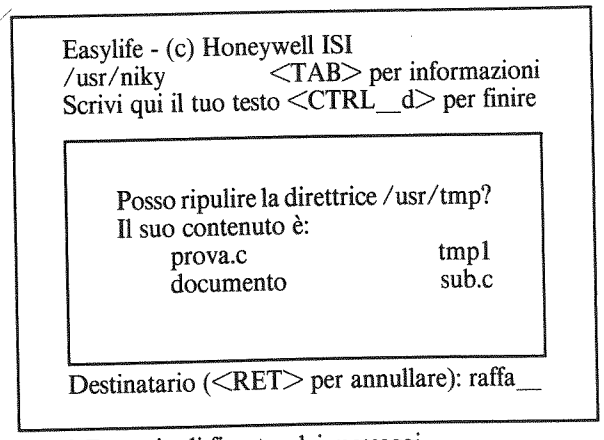


Fig. 2 Esempio di finestra dei messaggi.

7. METODOLOGIA DI DIALOGO

Come abbiamo già anticipato la tecnica di dialogo che qui si propone è un colloquio diretto che avviene attraverso la selezione di voci di menu contenuti in finestre.
In generale le azioni che un utente deve fare per poter lavorare col sistema sono le seguenti:

— muoversi all'interno del menu o della finestra degli oggetti;
— posizionarsi sull'azione o oggetto prescelti;
— premere il tasto di selezione <RETURN>.

7.1 I menu

Un menu è composto da un titolo e da un insieme di funzioni scritte in linguaggio naturale.
Le finestre che contengono i menu hanno dimensione variabile per quanto riguarda la lunghezza in relazione alla dimensione del menu. Se le voci in un menu sono superiori ad un limite massimo predefinito esiste un meccanismo di **scroll circolare** che permette di visualizzare tutto il contenuto della finestra e di ritornare automaticamente sulla prima voce di menu.

Dopo che un'azione è stata selezionata, si hanno tre diversi effetti:

— l'esecuzione diretta di un comando;
— la chiamata della finestra degli oggetti o la richiesta esplicita di un oggetto che rappresenta l'argomento del comando e la successiva esecuzione;
— la chiamata di un sottomenu.

Nel caso della chiamata di un nuovo menu, questo non si sovrappone completamente ai precedenti, ma lascia scoperto il titolo e il margine sinistro in modo da dare l'illusione di fogli di carta sovrapposti.

Quando ci sono molti menu in sequenza è possibile che si abbiano problemi di "overflow" su video. Esiste anche in questo caso un meccanismo di scroll che "shifta" i menu verso l'alto per fare posto a quelli selezionati.

Da ogni menu è possibile ritornare al precedente digitando il tasto <p>, o il tasto funzione <F1>. La posizione di ritorno è quella della funzione precedentemente selezionata.

L'implementazione grafica attraverso menu richiede all'utente poco sforzo mnemonico, inoltre permette la divisione delle azioni in gruppi distinti garantendo un disegno ordinato e una visione d'insieme completa.
Nel caso di utenti che acquistano dimestichezza nell'uso di Easylife, può però risultare noioso selezionare ogni volta un certo numero di menu per giungere ad una certa funzione, mentre è molto comodo avere alcune "scorciatoie" che permettono di raggiungere direttamente il menu desiderato.

Una di queste "scorciatoie" è stata implementata per il menu dei servizi il quale è il più usato soprattutto perchè permette la gestione degli errori in qualunque menu si

trovi, l'utente può andare direttamente al menu dei servizi semplicemente premendo il tasto <.>.
Un'altra scorciatoia è il cambio direttrice. È abbastanza frequente cambiare direttrice di lavoro ed è utile poterlo fare senza bisogno ogni volta di andare nel menu "gestione direttrici". Il tasto di </> permette di fare questo in qualunque menu ci si trovi.
Anche per quanto riguarda gli oggetti, specie se questi sono molti, sono possibili delle semplificazioni: il tasto <#> permette di specificare un particolare oggetto senza il bisogno di ricercarlo nella lista proposta.

8. A CHI È INDIRIZZATA?

Easylife è indirizzata innanzitutto all'utente inesperto del sistema Unix, il quale potrebbe essere sia l'utente finale sia il programmatore, abituato a vedere un linguaggio e non le funzionalità di un sistema.
Inoltre essa costituisce un comodo strumento per il "system administrator" o comunque per un utente esperto che voglia eseguire, in modo semplice e veloce, alcune operazioni complesse (creazione di un utente, personalizzazione delle stampanti) senza dover consultare i manuali.
Poiché ognuno di questi utenti ha esigenze diverse Easylife si propone a chiunque fornendo personalizzazioni diverse e mantenendo una rappresentazione visiva unica per tutti.

9. PERSONALIZZAZIONE

Personalizzazione in Easylife significa avere la possibilità di costruire un'interfaccia "ad hoc" per ogni utente.
È possibile quindi scegliere quali funzioni sono necessarie per un certo utente, e tramite una routine che è appunto "personalizzazione dei menu", fornire il set di funzioni stabilito.
Per fare alcuni esempi, un programmatore avrà tutte le funzioni relative al menu "produzione di software", l'amministratore del sistema avrà quelle relative al file system, l'utente finale avrà i menu dei servizi e le applicazioni (fig. 3).

Un altro aspetto della personalizzazione è la **gestione delle lingue nazionali,** ossia è possibile avere versioni di Easylife per ogni lingua utente.

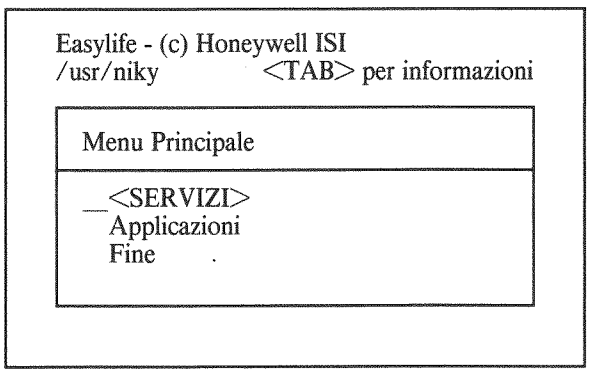


Fig. 3 Esempio di menu per l'utente finale.

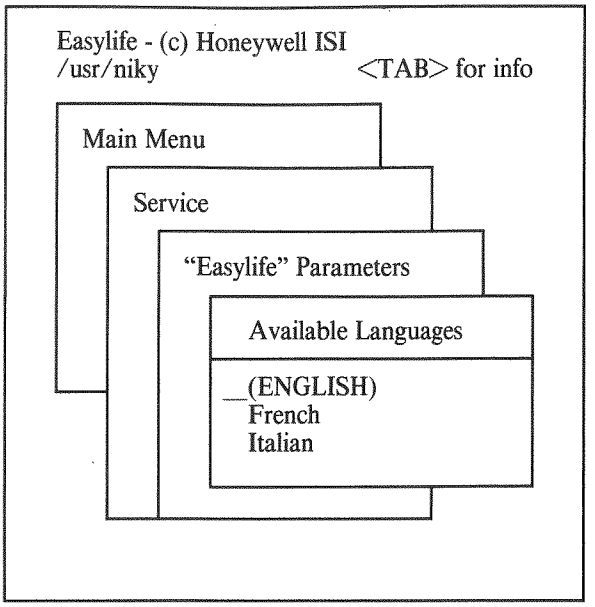


Fig. 4 Selezione della lingua desiderata.

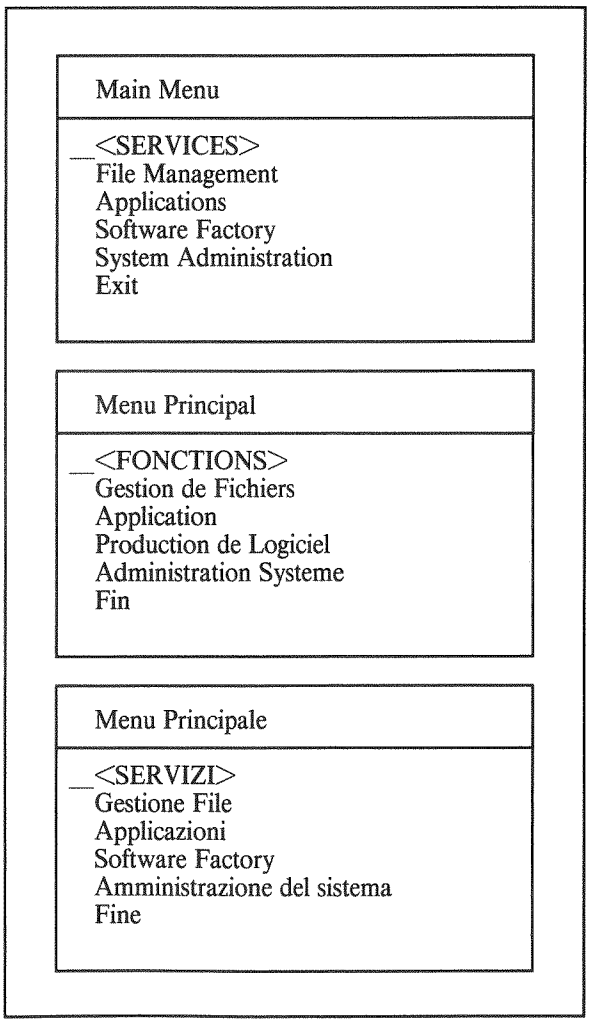


Fig. 5 Esempio di menu principale nelle lingue proposte.

Il proporre come lingua solo quella inglese, anche se questa è diventata una standard nel mondo informatico, non corrisponde al disegno di interfaccia orientata all'utente, infatti non tutti gli utenti conoscono l'inglese e comunque questo richiederebbe uno sforzo ulteriore nel comprendere le risposte del sistema.

In Easylife sono presenti tre lingue nazionali: Italiano, Inglese, Francese, richiamabili tramite selezione. Per le altre lingue viene fornita una routine che permette di tradurre tutto ciò che serve per il colloquio (menu, messaggi, help) nella lingua nazionale prescelta. Tale lingua viene poi aggiunta nel menu "Lingua utente".
L'utente, avendo a disposizione i menu in più lingue, può, in qualsiasi momento, cambiare la lingua con la quale colloquiare con il sistema semplicemente selezionandola (fig. 4).
È da notare che sullo stesso sistema si possono avere contemporaneamente versioni di Easylife diverse sia per le personalizzazioni dei menu sia per la lingua scelta, il tutto ottenibile con il minimo sforzo.

10. CARATTERISTICHE DEL PRODOTTO

Le principali caratteristiche di Easylife sono quindi:

— Facilità d'uso;
— Flessibilità;
— Gestione degli errori;
— Portabilità.

10.1 Facilità d'uso

La facilità d'uso è il principale obiettivo di Easylife; lo scopo è quello di fare sì che l'utente non sia in difficoltà nel compiere qualsiasi azione.
Entrando in Easylife si propongono all'utente tre componenti essenziali: il menu iniziale, la posizione nel file system e un insieme di informazioni di help.

Il menu iniziale, secondo la personalizzazione, contiene gli insiemi primari in cui muoversi. Così l'utente , entrando nel sistema con l'intento di programmare, selezionerà la voce produzione di software oppure, se vuole usare un pacchetto applicativo selezionerà la voce applicazioni e così via.

Il dialogo con il sistema avviene quindi semplicemente attraverso la selezione di voci di menu. Uniche eccezioni sono i casi in cui viene richiesta conferma su un'operazione selezionata o viene notificata la corretta esecuzione di una funzione. In queste circostanze i messaggi compaiono sull'ultima riga dello schermo.

La posizione nel file system è costituita dalla directory corrente, sempre visibile all'utente in modo che in ogni istante sappia sempre dove si trova.
Un messaggio **<TAB> per informazioni,** sempre presente nella seconda riga dello schermo, sta ad indicare che digitando il tasto <TAB> viene visualizzata una finestra

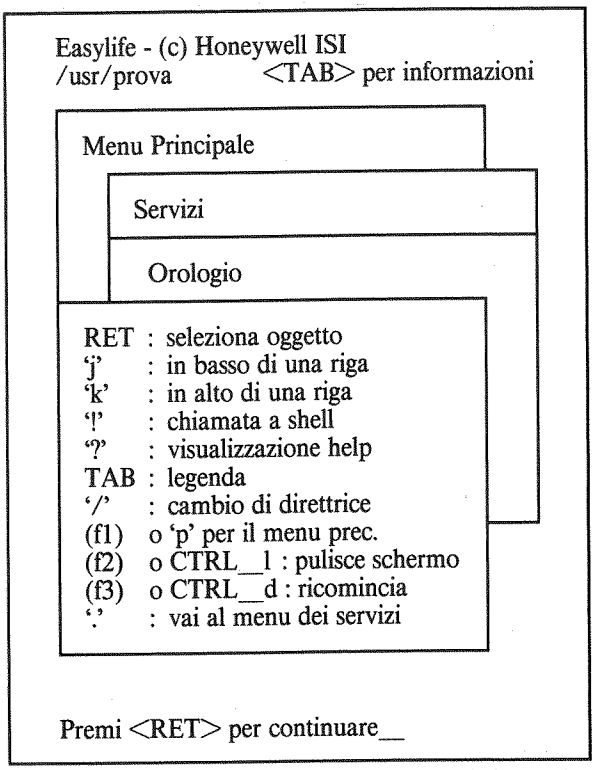


Fig. 6 Esempio di finestra di help relativa alle informazioni sul contesto.

contenente le informazioni necessarie per muoversi nelle finestre per selezionare funzioni di menu o oggetti (fig. 6).
I menu sono messi sullo schermo in modo che non siano mai completamente sovrapposti, così si conosce sempre la strada di accesso ad un menu.
L'utente non dovrebbe avere problemi nell'usare Easylife e comunque è sempre possibile selezionare un **help** con il tasto <?>.
Ci sono help relativi ad un particolare menu e help relativi ad ogni voce del menu in modo che ogni cosa sia sempre reperibile in modo dettagliato (fig. 7).

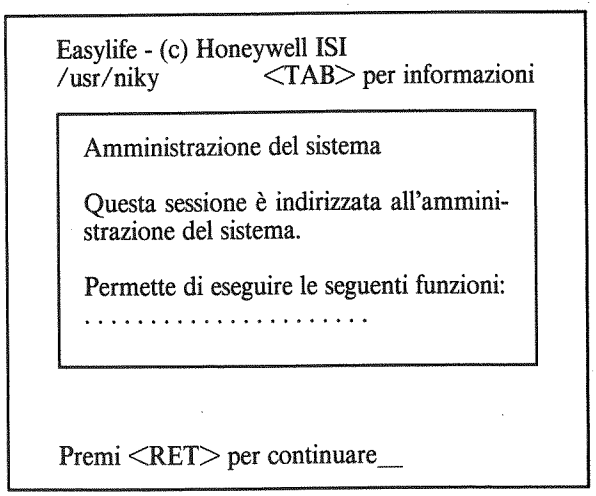


Fig. 7 Esempio di finestra di help relativa all'entrata del menu amministrazione del sistema

10.2 Flessibilità

La personalizzazione di cui abbiamo parlato è un aspetto importante che rende Easylife flessibile, infatti propone interfacce diverse ad utenti diversi.
Un altro strumento utile è il comando di **shell escape,** ossia con il tasto <!> è possibile abbandonare Easylife ed entrare nella shell di sistema. Questo è utile per gli utenti che desiderano eseguire compiti non previsti in Easylife oppure comandi ormai acquisiti. Con il tasto <CTRL_d> è possibile ritornare in Easylife ritrovando la stessa situazione di lavoro precedentemente lasciata.

Questo strumento è un modo comodo e veloce per utilizzare Easylife o la shell di sistema intercambiandoli a seconda delle necessità.

Esiste poi anche la possibilità di costruire un proprio menu, il cosidetto **menu utente.** in questo menu un utente può inserire comandi di sistema, nuovi comandi da lui stesso implementati, il richiamo di pacchetti, ecc. ... La costruzione del menu è guidata da Easylife che successivamente lo compila traducendolo in un formato compatibile con il resto del prodotto. Questo menu è richiamabile selezionando la voce "utente" (fig. 8).

Altre piccole caratteristiche che incidono sulla flessibilità sono la possibilità di definire un insieme di parametri come la lunghezza della pagina di stampa, il tipo di shell si sistema (shell, c-shell), il tipo di editor (vi, ed), il formato delle finestre, ecc...
L'utente così ha un;ampia possibilità di scelte in modo da avere un'interfaccia il più vicino possibile alle proprie esigenze.

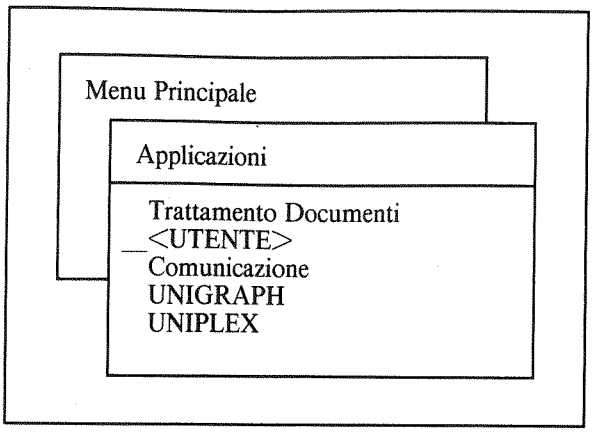


Fig. 8 Come selezionare il menu precedente

10.3 Gestione degli errori

I comandi Unix mandati in esecuzione da una funzione spesso ritornano dei messaggi di errore su standard error, Easylife ridirige questo file in un file particolare che può essere visualizzato in ogni momento da parte dell'utente. Tutto questo viene fatto perchè, altrimenti, i messaggi apparirebbero in zone qualsiasi dello schermo sovrapponendosi alle finestre esistenti e inoltre non sarebbe possibile avere una "storia" degli errori verificatisi
Si avvisa l'utente della presenza di un errore con un messaggio di **ERR** nella prima riga della schermo (fig. 9). L'utente ha la possibilità di visualizzare o rimuovere questo file tramite la funzione "gestione errori" (fig. 10).
Inoltre ogni utente Easylife possiede un file errori distinto da quello di altri utenti. In questo file sono presenti, per

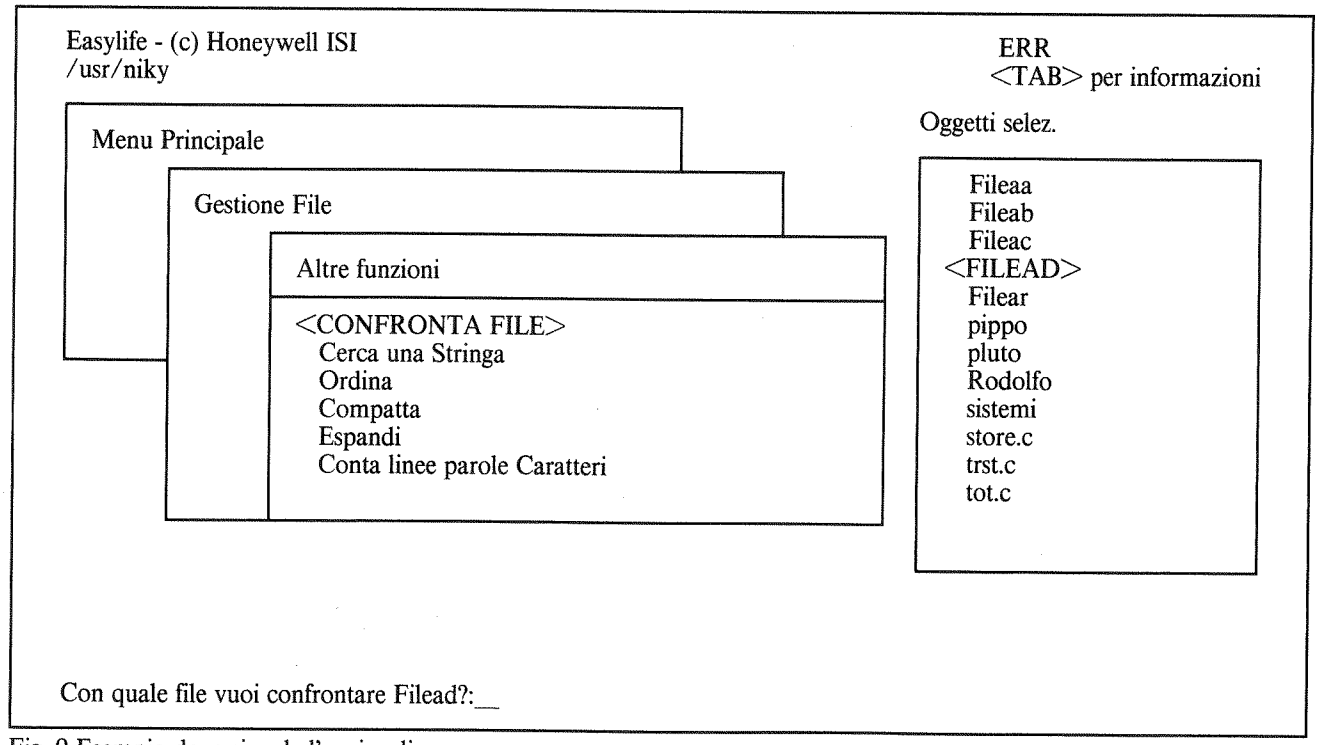


Fig. 9 Esempio dove si vede l'avviso di errore

ogni errore, il messaggio relativo seguito dal nome della funzione che lo ha generato, dalla data e dall'ora di generazione in modo che sia di facile comprensione (fig. 11).

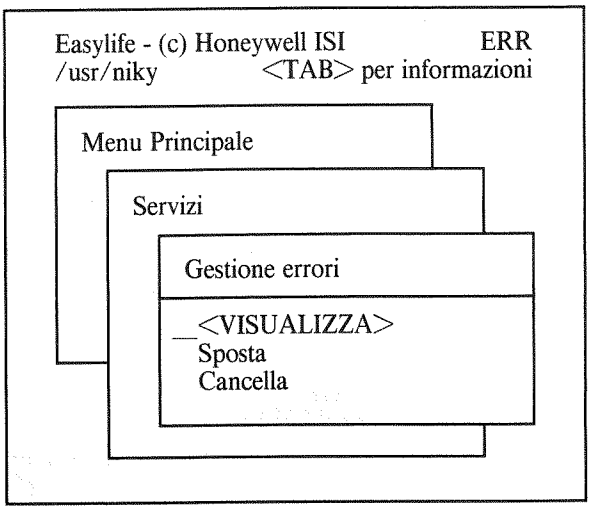


Fig. 10 Menu per la gestione degli errori

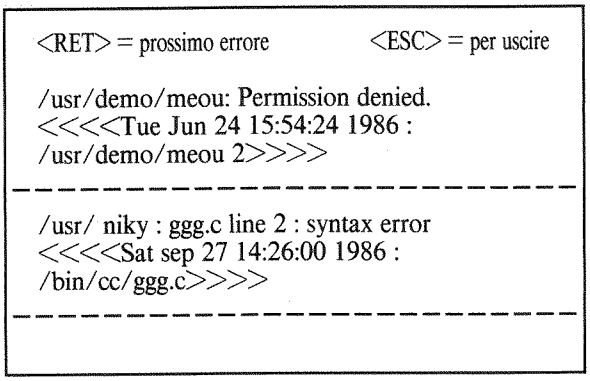


Fig. 11 Esempio di finestra relativa alla gestione degli errori

10.4 Portabilità: strumenti di input e output

Easylife, essendo un'interfaccia per il sistema Unix, deve essere portabile su diversi tipi di macchine.
Questo è realizzabile in quanto si è scelto come linguaggio di implementazione il C-language, che è indipendente dall'hardware della macchina. Ogni macchina poi supporta diversi tipi di terminali per cui è necessario che Easylife venga implementata in modo da essere idipendente dalle periferiche utilizzate.

Il tipo di terminale che si considera come modello è un terminale con il minimo di attributi visivi e con un display non sofisticato.
Non si considera nessun altro strumento di I/O come il mouse, tavolette grafiche o altro.
Per evidenziare il menu corrente, per le cornici delle altre finestre e per evidenziare la voce su cui si è posizionati, si utilizza l'attributo di **inverse-video.**

Dove non c'è l'attributo di inverse-video, la posizione corrente viene evidenziata portando i caratteri da minuscolo a maiuscolo e chiudendo la voce tra i caratteri "<>".
L'utente ha comunque la possibilità, su alcuni terminali, di scegliere il formato delle finestre. C'è infatti un menu "formato delle finestre" nel quale è possibile selezionare il tipo di finestra più idoneo: con l'inverse-video totale, parziale (cioè solo sul titolo del menu corrente) o senza inverse-video.

Per digitare e per muoversi nelle finestre si usa la tastiera. Sempre per motivi di portabilità comunque vengono supportati i tasti funzione, i tasti di posizionamento del cursore e i tasti corrispondenti a semplici caratteri ASCII in modo da coprire anche i terminali meno sofisticati.
Il risultato di ciò è un prodotto che riconosce automaticamente il tipo di terminale su cui viene eseguito e si comporta di conseguenza, senza bisogno di toccare il codice, e che può essere reso portabile su molti altri tipi con il minimo intervento (figg. 12, 13).

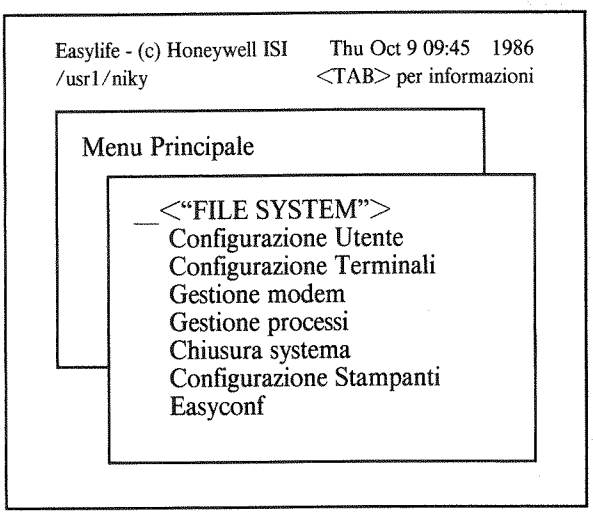


Fig. 12 Esempio di videata relativa ad un terminale non grafico senza l'attributo di inverse-video

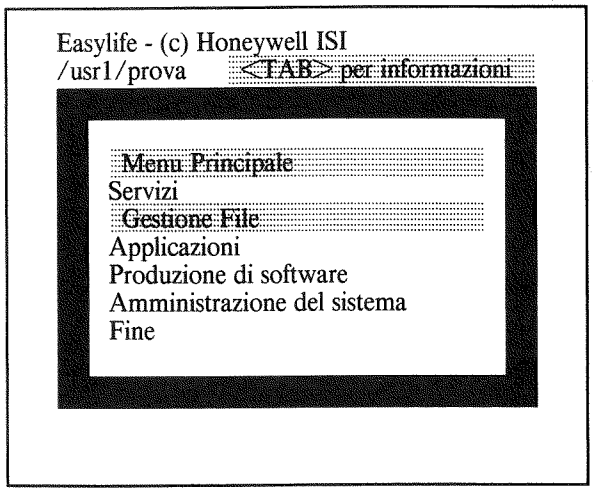


Fig. 13 Esempio di videata relativa ad un terminale non grafico con l'attributo di inverse-video

11. PRESENTATION

Easylife si presenta con tre tipi diversi di finestre: la finestra contenente i menu (dei quali abbiamo già parlato), la finestra contenente gli oggetti, la finestra degli help. La rappresentazione metaforica vuole esere quella dei fogli sulla scrivania.

11.1 Finestra oggetti

La finestra in questione contiene una lista di oggetti in ordine alfabetico relativi all'azione selezionata.
Essa è completamente distinta dai menu e non si sovrappone a loro in modo da avere sempre visibili le azioni e gli oggetti corrispondenti.
Anche questa finestra varia a seconda del numero degli oggetti fino ad un massimo di diciassette, dopo di che, anche in questo caso, è previsto il meccanismo di scroll circolare.

11.2 Finestre di help

Come precedentemente descritto abbiamo due tipi di tali finestre di help.
Una è selezionabile con il tasto <**TAB**> e contiene le **informazioni sul contesto,** cioè le descrizioni dei tasti funzione i quali permettono sia il movimento sia la selezione nei menu e nella finestra degli oggetti.
Queste informazioni cambiano ogni volta in dipendenza dal contesto in cui ci si trova, mantenendo una relazione fissa tra azione e tasto che la esegue (figg. 14, 15).

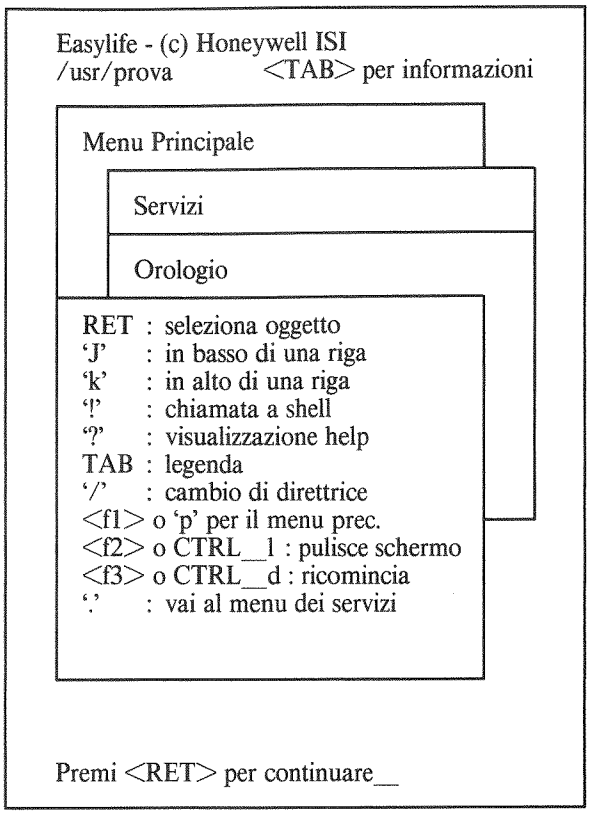


Fig. 14 Esempio di finestra di help relativa alle informazioni per le finestre dei menu

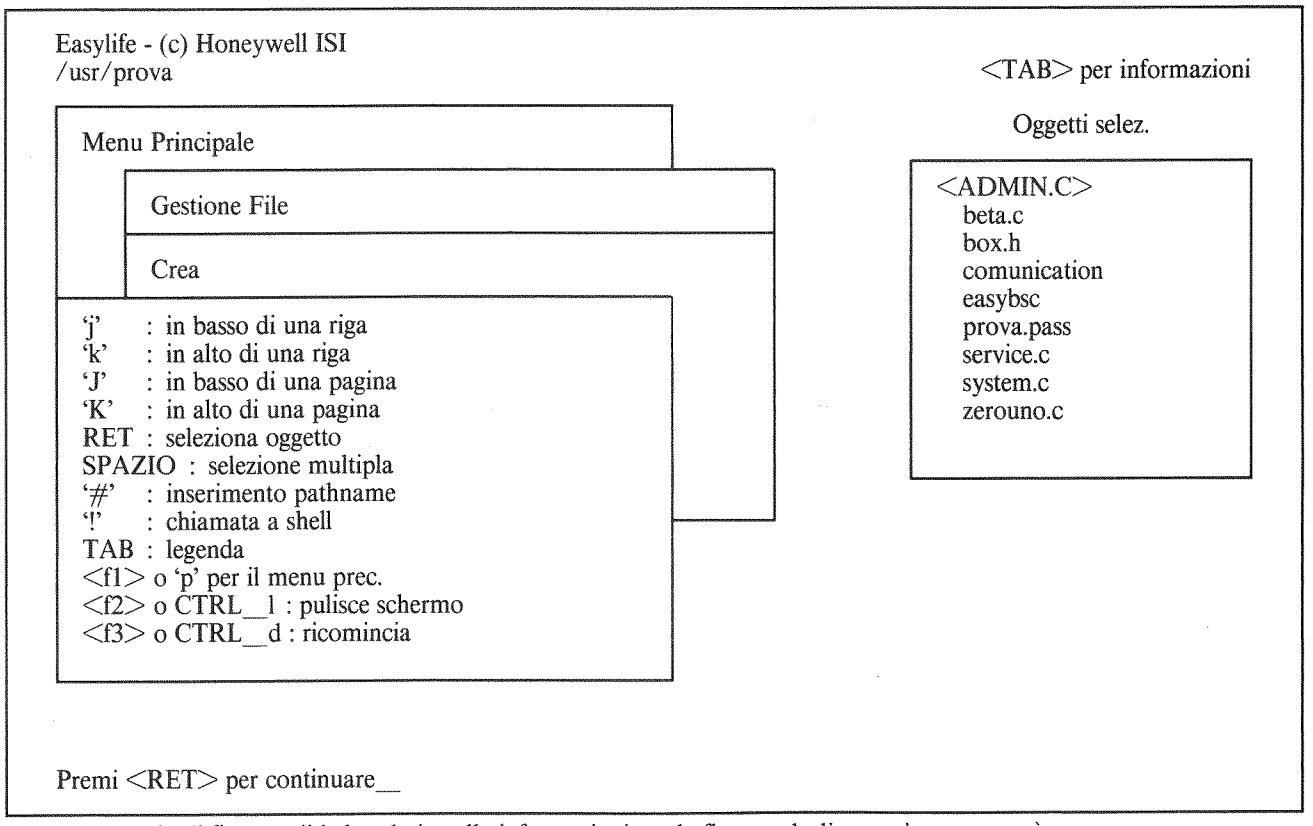


Fig. 15 Esempio di finestra di help relativa alle informazioni per la finestra degli oggetti

Un altro tipo di finestra di help contiene invece informazioni relative ad una voce del menu ed è selezionabile con il tasto <?>. Per poterla richiamare è necessario prima posizionarsi sulla voce di cui si vuole l'informazione (fig. 6).

11.3 Ambiente di lavoro

L'ambiente di lavoro di un utente Easylife è costituito dalla propria Home-Directory e da tutte quelle ad essa sottostanti. Egli ha visibilità immediata degli oggetti presenti nella directory corrente ed è necessario cambiare direttrice per avere visione degli altri oggetti. Inoltre, ad un utente easylife non viene concesso dieseguire azioni su file non di sua proprietà.

12. POSSIBILI ESTENSIONI FUTURE

Si sono pensati altri tipi di applicazioni che possono essere sviluppate in un immediato futuro e che costituiranno Easylife+.
Un'idea è quella di trasformare Easylife in uno strumento didattico: per ogni funzione selezionata si potrebbe far apparire sul video un messaggio che specifichi il comando Unix da eseguire, le opzioni e gli argomenti relativi. Questo potrebbe essere un modo semplice e divertente per imparare ad usare il sistema.

Un'altra possibile estensione sarebbe quella di fornire delle maschere per ogni funzione sulle queli siano specificate le opzioni relative al comando in questione; l'utente potrebbe allora selezionare le opzioni necessarie alla propria applicazione. Queste maschere dovrebbero però essere opzionali e richiamabili a richiesta, in quanto non necessarie per un utente inesperto.
Inoltre si pensa di integrare Easylife in modo che gestisca i pacchetti di "comunication" applicabili sullo specifico sistema operativo.

13. RINGRAZIAMENTI

Il lavoro è stato realizzato nei laboratori Honeywell di Pregnana Milanese. Si ringraziano tutti coloro che hanno collaborato alla realizzazione del prodotto.

14. BIBLIOGRAFIA

[1] Carrol J.M., PRESENTATION AND FORM IN USER-INTERFACE ARCHITECTURE, Byte, Dec. 1983, pag. 263, 280

[2] Dehning W., Essig H., Maass S., THE ADAPTATION OF VIRTUAL MAN-COMPUTER INTERFACES TO USER REQUIREMENT IN DIALOGS, Berlin, Springer-Verlag, 1981

[3] Haring G., Krasser T., A RELATION OF A HUMAN-COMPUTER INTERFACE FOR NAIVE-USERS - A CASE STUDY, from "Readings on Cognitive Ergonomics-Mind and Computer", Berlin, Springer-Verlag, 1982, pag. 182, 190

[4] Herbach M., Katz R., Landau J., THE USER INTERFACE: TWO APPROACHES, Byte, Dec. 1983, pag. 247, 259

[5] Lee A., Lochovsky F.H., USER INTERFACE DESIGN, from "Office Automation", ed. by Tsichritzis, pag. 3, 20, Berlin, Springer-Verlag, 1981

[6] Lieberman H., DESIGNING INTERACTIVE SYSTEMS FROM THE USER'S VIEWPOINT, from "Integrated Interactive Computing Systems", Degano P. & Sandewall, (ed), North-Holland Publishing Company E-CICS, 1983

[7] Pope A., Kates G., Fineberg D., MAKING LIFE EASIER FOR PROFESSIONAL AND NOVICE PROGRAMMERS, Byte, Dec. 1983, pag. 155, 160

[8] Quartiroli I., Fusaro M., Smareglia S., UNIX INTRODUZIONE AL SISTEMA OPERATIVO, Clup, Milano, 1983

[9] Rebecca Thomas PhD, Yates J., A USER GUIDE TO THE UNIX SYSTEM, Publ. by Osborne/McGraw-Hill Berkeley, California, 1983

[10] Rohr G., Tauber M.J., REPRESENTATIONAL FRAMEWORKS AND MODELS FOR HUMAN-COMPUTER INTERFACES, from "Readings on Cognitive Ergonomics-Mind and Computer", Berlin, Springer-Verlag, 1984, pag. 127, 128

[11] Smith D.C., Irby C., Kimball R., Verplank B., Harslem E., DESIGNING THE STAR USER INTERFACE, Byte, vol. 7, n. 4, Apr. 1982, pag. 242, 281

[12] Vandor S., THE STARBURST USER INTERFACE, Byte, Dec. 1983, pag. 189, 193

[13] Warfield R.W., THE NEW INTERFACE TECHNOLOGY. AN INTRODUCTION TO WINDOWS AND MICE, Byte, Dec. 1983, pag. 218, 230

[14] Manuale Uniplus System V

SCIENZA DEL SOFTWARE: METODI E RISULTATI

Leonardo Felician, Graziella Zalateu

Istituto di Matematica, Informatica e Sistemistica

Università di Udine

RIASSUNTO

Oggetto di questo lavoro è la presentazione di una teoria denominata "Software Science" o "Scienza del Software", che si propone di misurare alcuni parametri intrinseci dei programmi e di costruire da essi alcune valutazioni sulla "qualità" dei programmi stessi.
Questo modo di procedere è stato applicato ad un campione comprendente tutti i programmi in linguaggio Pascal disponibili presso il Centro di Calcolo dell'Università di Udine, costituendo essi una delle più estese verifiche sperimentali della teoria stessa, comunque la più completa per quanto riguarda il linguaggio Pascal.

ABSTRACT

This paper introduces a theory named "Software Science" which states the goal of measuring some parameters which are characteristics of the programs. Some evaluations on the "quality" of the programs are achievable from these results.
These procedure has been applied to a sample concerning all the programs written in Pascal laguage at the Computer Center of the University of Udine. These results represent one of the most wide experimental checks of the whole theory, and surely the most complete as far as Pascal language programs are concerned.

1. INTRODUZIONE

Lo studio analitico di alcuni parametri intrinseci dei programmi è oggetto di ricerca a partire dagli anno '70, quando, a causa del mutamento di dimensioni e di caratteristiche delle applicazioni software, (che si trasformavano da procedure scollegate a complessi progetti di sistemi informativi) si cominciò a porre sempre maggior attenzione da un lato agli aspetti di gestione di un processo E.D., dall'altro alle caratteristiche proprie della scrittura dei programmi.

La prima branca di interesse, che considera il fenomeno software dal punto di vista macroscopico, diede origine alla teoria dei sistemi informativi aziendali, alle tecniche di analisi e programmazione strutturata e alle metodologie di sviluppo e conduzione di grossi progetti E.D., sottolineando la necessità di un approccio coordinato a queste problematiche e riconoscendo per la prima volta che un insieme di programmi non bastano per costruire un sistema informativo.

Il secondo filone di ricerca, rivolto invece agli aspetti microscopici, ha portato alla definizione della "ingegneria del software" e più in generale al tentativo di fornire delle definizioni operative di qualità misurabili dei programmi.

Rientra in questo secondo campo d'indagine la teoria denominata "Scienza del Software" o "Software Science" sviluppata originariamente da M. H. Halstead per il linguaggio Fortran e ripresa successivamente da altri autori. Secondo questo approccio è possibile individuare alcune proprietà intrinseche dei programmi a partire dalla misura di poche grandezze fondamentali, che possono essere tra l'altro misurate tramite semplici algoritmi. A partire da queste variabili di base si possono costruire altre grandezze variamente correlate, dotate di significato autonomo e vicine ad indentificare proprietà importanti dei programmi, quali il contenuto intellettivo e lo sforzo di programmazione.

Dopo una panoramica dei fondamenti della Scienza del

Software, completa di verifiche e critiche apportate da numerosi ricercatori, questo lavoro si rivolge ad un'analisi sperimentale delle relazioni fondamentali eseguita tramite un algoritmo proprio accuratamente calibrato per tener conto di tutte le anomalie riconoscibili nei programmi.

L'analisi è stata condotta su tutti i programmi in linguaggio Pascal disponibili nelle librerie del Centro di Calcolo dell'Università di Udine, ottenendo così dei risultati significativi dal punto di vista statistico e di interesse generale, poiche per la prima volta questa teoria è stata sottoposta ad una verifica così ampia, almeno per quanto riguarda il linguaggio Pascal.

La conclusione di questo lavoro dunque, positiva per quanto riguarda i risultati della ricerca, evidenzia tuttavia i limiti di questo approccio al problema, insiti nel fatto che per poter lavorare su un gran numero di programmi è indispensabile limitarsi a guardare agli stessi da un punto di vista macroscopico, e cioè senza entrare nel dettaglio della loro struttura.

Con questa doverosa limitazione di obiettivi, la Scienza del Sofware risulta dunque un punto di contatto tra un modello teorico di grande semplicità e potenza e la pratica di una realtà applicativa sempre più profondamente basata sulla produzione su larga scala del software.

2. RELAZIONI TRA LE VARIABILI FONDAMENTALI

Maurice H. Halstead propose, nel '72, una teoria generale denominata "Software Science" o "scienza del Software" [1], che riguarda solo le proprietà degli algoritmi che possono essere misurate direttamente o indirettamente, staticamente o dinamicamente, e le relazioni fra queste proprietà che rimangono invariate nel passaggio da un linguaggio ad un altro.

L'idea di base della teoria di Halstead è la seguente: data una implementazione dell'algoritmo, in un qualsiasi linguaggio, è possibile identificare tutti gli operandi, definiti come le variabili e le costanti usate. Analogamente è possibile identificare tutti gli operatori, definiti come simboli o combinazioni di simboli che toccano il valore o l'ordine di un operando (cfr. tabella 1).

Operatori =	parole chiave del linguaggio e segni di punteggiatura
Operandi =	variabili e costanti

Tabella 1 Definizione di operatori ed operandi

Dall'identificazione di operatori ed operandi si può definire un certo numero di entità misurabili che devono essere presenti in ogni versione dell'algoritmo. Queste proprietà sono le metriche base dalle quali sono state ottenute le relazioni della Scienza del Software. Le proprietà suddette sono:

ETA1	:	il numero di operatori unici usati nell'implementazione
ETA2	:	il numero di operandi unici usati nell'implementazione
N1	:	il numero di operatori totali
N2	:	il numero di operandi totali

Tabella 2 Significato dei parametri fondamentali alla base della metrica

Da queste grandezze di base è opportuno definire il vocabolario

$$ETA = ETA1 + ETA2,$$

inteso come l'insieme delle parole necessarie per la scrittura del programma, e la lunghezza dell'implementazione

$$N = N1 + N2$$

definita come l'insieme di tutti gli operatori ed operandi usati nell'implementazione.

Il concetto che un algoritmo consista di operatori ed operandi e nient'altro, è più facilmente comprensibile considerando linguaggi vicini al linguaggio macchina il cui formato d'istruzione è costituito da sole due parti: il codice dell'operazione e l'indirizzo dell'operando. Bisogna infine notare che la suddivisione in operatori ed operandi è soggetta sia al tipo di linguaggio sia a convenzioni usate dallo sperimentatore, le quali, ovviamente, devono essere chiaramente definite e dichiarate.

3. LA LUNGHEZZA DI UN PROGRAMMA

La prima e più importante relazione che si può trovare fra i parametri precedentemente definiti, è una relazione di tipo quantitativo fra la lunghezza N e il vocabolario ETA. Una stringa di lunghezza N, costruita con gli elementi di un vocabolario di ETA simboli, deve obbedire ad un certo numero di limitazioni. Ovviamente la condizione che ognuno degli ETA simboli debba apparire almeno una volta garantisce che

$$ETA <= N$$

cosicchè esiste un limite inferiore a N in termini di ETA. D'altra parte aggiungendo un'altra condizione si troverà anche un limite superiore secondo il seguente procedimento:

— si suddivide la stringa di lunghezza N in sottostringhe di lunghezza ETA;
— così diviso, un programma consisterà di N/ETA istruzioni ognuna di lunghezza ETA;
— imponendo ora che una stringa non contenga due

sottostringhe uguali di lunghezza ETA, si ottiene anche il limite superiore.

Questa condizione è molto ragionevole nei programmi in cui l'economia di espressioni richiede che venga assegnato un nome distinto ad un'espressione comune, dove con espressione si intende un'espressione fra più variabili, in modo che essa venga valutata una volta sola. Di conseguenza, se una sottoespressione comune viene utilizzata in un programma più di una volta, essendo assegnata ad un unico operando, essa incrementerà ETA soltanto di una unità. Il munero di possibili combinazioni di ETA simboli presi ETA alla volta ETA^{ETA}, perciò il limite superiore cercato è

$$N <= ETA^{ETA} \qquad (1)$$

Questa stima può essere ulteriormente migliorata notando che il vocabolario consiste sia di operatori che di operandi e che essi tendono ad alternarsi. Infatti, se si considera un programma costituito da due operatori e da due operndi, il numero delle possibili stringhe di quattro simboli presi quattro alla volta è $4^4 = 256$. Questo numero comprende anche le stringhe costituite solo da operatori o solo da operandi, combinazioni che non sono ammissibili, poiché non si troverà mai un programma costituito solo da operatori o da operandi. Le stringhe possibili, presentate nella tabella 3 relativamente al semplice esempio citato, sono in numero notevolmente inferiore a quello previsto.

(1) AaAa	(5) AaAb	(9) AbAa	(13) AbAb
(2) AaBa	(6) AaBb	(10) AbBa	(14) AbBb
(3) BaAa	(7) BaAb	(11) BbAa	(15) BbAb
(4) BaBa	(8) BaBb	(12) BbBa	(16) BbBb

Tabella 3 Insieme delle possibili combinazioni di due operatori (A, B) e due operandi (a, b)

La considerazione suddetta porta a sostituire, come in Halstead [1], l'equazione (1) con

$$N <= ETA^{ETA} * ETA1^{ETA1} * ETA2^{ETA2}$$

Ora il limite superiore deve contenere non solo l'insieme ordinato di N elementi caratteristici del programma ma anche tutti i possibili sottoinsiemi dell'insieme ordinato. Il numero di possibili sottoinsiemi di un insieme ordinato di N elementi è 2. Quindi si può eguagliare il numero di possibili combinazioni di operatori ed operandi con il numero di elementi della famiglia dei sottoinsiemi ottenendo:

$$2^N = ETA1^{ETA1} * ETA2^{ETA2}$$

Dal quale si ricava

$$\hat{N} = ETA1 * (\log_2 ETA1) \ll ETA2 * (\log_2 ETA2)$$

$\hat{N}$ è, in definitiva, il numero di bit necessari per rappresentare, almeno una volta, tutti gli operatori ed operandi del programma. Infatti se, per esempio, un algoritmo è costituito da ETA1 operatori e da ETA2 operandi, il numero di bit necessari per indivuarli tutti distintamente è, rispettivamente, log ETA1 e log ETA2. Questi ultimi moltiplicati per il numero di operatori ed operandi unici, danno, infine, il numero di bit necessario per rappresentare i simboli della tabella almeno una volta.

Per poter avere carattere generale, una qualsiasi relazione derivata da considerazioni così qualitative deve esere sottoposta a molti test al fine di verificarne la validità. Halstead la applicò ad un insieme molto limitato di programmi in Fortran pubblicati in [2]. I suoi risultati mostrano una stretta correlazione fra N ed $\hat{N}$ e questo successo incoraggiò ulteriori ricerche in questo campo.

Haistead cercò, in sintesi, di trovare una formula che permettesse di calcolare la lunghezza di un programma noti solo il numero di operatori ed operandi unici. Questi sono dei parametri che si suppone possano essere già noti prima della stesura del programma, e cioè in fase di progettazione.

È evidente l'interesse per questo tipo di approccio: una teoria di questo tipo perfettamente funzionante potrebbe infatti offrire valutazioni quantitative al tempo richiesto per sviluppare un certo insieme di programmi, fornendo dunque un potente aiuto ai responsabili della conduzione di progetti informatici.

A partire dalla lunghezza del programma, comunque, si possono ricavare facilmente altri parametri, utili per fornire valutazioni esterne su alcune proprietà intrinseche del software, e cioè:

- il volume di un programma, che rappresenta il minimo numero di bit occupato da tutti gli operatori ed operandi dell'algoritmo;
- il livello di programmazione, che rivela sia la differenza di capacità fra programmatori che la differenza fra programmi di diversa dimensione;
- il livello del linguaggio, che rappresenta una costante propria di ogni linguaggio;
- l'"intelligence content", ovvero il contenuto intellettivo, che permette di dire se due programmi, implementati in linguaggi diversi, realizzando la stessa funzione;
- lo sforzo di programmazione, che fornisce in fase di progettazione una valutazione del tempo richiesto per implementare l'algoritmo.

4. CRITICHE E CONFERME ALLA TEORIA

Altri ricercatori si sono posti il problema di validare la relazione di Halstead con linguaggi diversi, con insiemi diversi di algoritmi o con algoritmi di lunghezza diversa: i risultati di questi studi sono esposti in questo capitolo. È comunque il caso di ricordare che gran parte dei test finora effettuati sono poco significativi dal punto di vista statistico, perchè si limitano a considerare un piccolo numero di programmi.

Tra i risultati più interessanti si può citare quello ottenuto dai ricercatori del Centro di Calcolo della Purdue University in California [3], i quali verificarono, nel '74, la formula con un insieme di 329 programmi scritti in Fortran. Essi trovarono un coefficiente di correlazione di 0.95 che saliva a 0.98 se si escludevano i programmi più voluminosi: questo è un primo avviso della discrepanza tra la formula Halstead e la realtà per i programmi grossi, che sarà chiarita nel seguito.

J.L. Elshoff [4], [5], dei Laboratori di Ricerca della General Motors, confermò la validità della formula per un grosso insieme di programmi in PL/1: anche il suo coefficiente di correlazione fu molto alto: 0.98.

Nel 1981 D. Johnson e A.M. Lister [6], del Department of Computer Science dell'Università di Queensland, affermarono che la formula trovata da Halstead non era valida per programmi scritti in linguaggi strutturati come il Pascal. Essi sostenevano che la formula per il calcolo della lunghezza, supportata da esperimenti con programmi particolari in Fortran ed in PL/1, non era valida senza modifiche per il linguaggio Pascal.

L'applicazione della suddetta formula a programmi in Pascal portò alla conclusione che l'equazione poteva essere valida per programmi corti, ma per quelli di una certa consistenza presentava forti discrepanze per cui $\hat{N}$ non poteva essere considerata in alcun modo una buona stima di N.

Al fine di cercare di capire il perchè di queste discrepanze, essi fecero la seguente analisi: innanzitutto notarono che $\hat{N}$ sottostimava considerevolmente N per i programmi considerati. Esaminando le misure riconobbero che per programmi grossi ETA1 era considerevolmente inferiore a ETA2, mentre per quelli piccoli essi erano di grandezza paragonabile. Da questa osservazione fu facile dedurre che la causa della discrepanza era lo scarso contributo dato da ETA1: nel calcolo delle variabili sono infatti molto importanti le convenzioni usate, che secondo Johnson e Lister sono le seguenti: essi consideravano "operatori":

1. gli operatori interni, le procedure e le funzioni del linguaggio;
2. le procedure e le funzioni definite dell'utente;
3. le label che sono etichette di un trasferimento di controllo.

Si sottolinea così che i programmi scritti in Pascal, o in un altro linguaggio strutturato consentono un numero molto piccolo di operatori della classe 3. Ciò significa che una volta che tali programmi sono sufficientemente grossi da contenere la maggior parte di operatori predefiniti, la crescita di ETA1 con il programma è limitata dal numero di procedure e funzioni definite dall'utente. Quindi il motivo della differenza è dovuto alla diversità delle strutture di controllo.

Si può allora suggerire di alterare il modo di contare del Pascal, cosicchè ogni occorrenza di una struttura di controllo sia contata come un operatore distinto, senza alcun riguardo al numero delle volte in cui viene usata.

La tabella 3 evidenzia il miglioramento ottenuto nella correlazione fra N ed N nella colonna identificata N1.

Numero dell' Algoritmo	$N/\hat{N}$	N/N1
(1)	2.28	1.37
(2)	2.41	1.36
(3)	1.48	0.822
(4)	1.12	0.655
(5)	1.72	1.06
(6)	1.20	0.895
(7)	1.64	0.779
(8)	1.61	1.08
(9)	1.57	0.863
Valore medio	1.67	0.987
Deviazione standard	0.408	0.208

Tabella 3 Lunghezza calcolata secondo Halstead ($\hat{N}$) e correzioni di Johnson e Lister (N1) [6]

Questa osservazione, cioè la non validità della formula di Halstead, sembra essere in contrasto con i risultati, precedentemente citati, ottenuti da Elshoff per programmi in PL/1 [4].
Johnson e Lister sottolienarono che il suo alto coefficiente di correlazione era dovuto sia all'uso del 'GOTO', nei programmi da lui considerati particolari per ottenere tale accordo.

Nel 1982, V.Y. Dhen, S.D. Conte e H.E. Dunsmore, [7] dimostrarono a loro volta l'imprecisione dell'equazione della lunghezza di Halstead. Essi proposero, come alternativa, la seguente:

$$NF = \log(ETA1!) + \log(ETA2!)$$

L'espressione riportata fu verificata con un insieme di 202 programmi Pascal e, accanto al valore della lunghezza di Halstead e della loro espressione, calcolarono le seguenti quantità:

$$dH = \frac{I\,N - \hat{N}\,I}{N} \qquad dF = \frac{I\,N - NF\,I}{N}$$

Si osservò che per i loro programmi la discrepanza esistente fra N ed $\hat{N}$ era molto più alta di quanto fosse stato dimostrato in studi passati.

5. DESCRIZIONE DELL'ALGORITMO DI MISURA

Per verificare le relazioni esposte è stato sviluppato un algoritmo per il calcolo dei parametri introdotti da Hal-

stead, successivamente applicato a 550 programmi significativi scritti in Pascal.

Si vuole, ancora una volta, precisare che il calcolo dei parametri dipende dall'esistenza di convenzioni ben fissate, le quali richiedono definizioni precise di operatori ed operandi. Buone descrizioni non implicano strategie corrette, ma assicurano che le strategie possano esere capite, testate e valutate. Molto spesso, infatti, si può notare che le convenzioni usate per il calcolo dei parametri sono mal poste o addirittura inesistenti. Ciò ovviamente impedisce la loro comprensione, la possibilità di dare una loro valutazione precisa ed il loro confronto con quelle di altri autori.

Dato che non c'è la necessità di comprendere le funzioni eseguite dai programmi analizzati, le spiegazioni sulla logica dei programmo ed i commenti al suo interno sono superflui. Infatti i commenti potrebbero impedire l'esatta valutazione del codice e togliere valore ai risultati ottenuti.

Nella scansione dei programmi si sono ignorate:

— le intestazioni dei programmi, incluse la parole chiave (PROGRAM, FUNCTION, PROCEDURE e MODULE);
— le parti dichiarative relative a: variabili, tipi, costanti, label ed i parametri locali delle procedure;
— i commenti.

I nomi delle procedure e delle funzioni che seguono le parole chiave PROCEDURE, FUNCTION, sono memorizzati in una tabella accanto agli operatori, in modo da poterli localizzare nel caso di una chiamata (a procedura o a funzione).

I segni di punteggiatura sono stati suddivisi in tre categorie particolari a causa della possibilità del presentarsi di coppie di essi. Nel primo gruppo sono stati memorizzati i seguenti simboli che possono preludere ad un'eventuale coppia:

' : '	doppio punto	per ' := '		
' > '	maggiore	per '>= '		
' < '	minore	per '<= '	oppure	per '<>'
' * '	asterisco	per ' *) '	oppure	per ' ** '

Nel secondo gruppo sono compresi i segni di punteggiatura che potrebbero presentarsi in coppia con i precedenti:

'>'	maggiore
'='	uguale
')'	parentesi chiusa
'*'	asterisco

Nel terzo sono memorizzati i rimanenti simboli che, come precedentemente stabilito, sono considerati operatori.

Si vuole infine presentare le condizioni richieste ai programmi presi in considerazione in questo lavoro:

1. Ogni programma, prima di essere dato in input all'algoritmo che ne calcola i parametri fondamentali, è stato compilato al fine di eliminare i programmi sintatticamente scorretti. Essi infatti altererebbero i risultati in modo determinante.
2. Dopo aver testato tutti i programmi, si è controllato con cura che non ci fossero programmi identici, cioè con valori dei parametri fondamentali uguali o molto simili, che alterano una qualsiasi valutazione statistica che si possa compiere su di essi.
3. Sono stati presi in esame tutti i programmi in Pascal disponibili nelle librerie dell'elaboratore DIGITAL VAX 11/780 del Centro di Calcolo dell'Università degli Studi di Udine.

6. RISULTATI SULLA LUNGHEZZA DEI PROGRAMMI

Il programma descritto ha permesso di determinare i parametri fondamentali dai quali si può ricavare la lunghezza di un programma intesa come somma di operatori ed operandi totali:

$$N = N1+N2 \qquad (1)$$

Se si esprime la lunghezza in funzione del vocabolario, si può osservare che essa presenta un andamento ben preciso. Questo è messo in evidenza, in figura 1, dalla curva polinomiale che meglio approssima tali punti, rappresentati qui inscala logaritmica. Halstead cercò di stimare tale andamento con la formula presentata in precedenza.

Risulta evidente dalla Tabella 4, che dà i valori medi di N ed N per le varie classi di programmi, ed ancora meglio dalla figura, che N è una sovrastima di N per i valori piccoli di ETA, mentre ne è un'evidente sottostima per valori elevati del vocabolario. Non deve stupire la stretta correlazione trovata da Halstead sui programmi da lui esaminati, in antitesi con i risultati qui esposti: semplicemente egli ha dimostrato la validità della formula per un insieme limitato di programmi, quasi tutti della stessa lunghezza, scritti in Fortran.

Shen, Conte e Dunsmore, accorgendosi evidentemente della sovrastima, proposero la formula descritta al paragrafo 3. Anche quest'ultima è stata applicata ai parametri fondamentali qui ricavati. Ancora una volta, dalla tabella 4 e dalla figura, si può osservare che, se per valori piccoli di ETA, la sovrastima è inferiore, per valori elevati la sottostima è ancora peggiore. Ciò è del resto in accordo con i dati da essi ricavati.

Seguendo il suggerimento di Johnson e Lister, i quali si soffermarono sul perchè la formula suggerita da Halstead non si adattasse ai dati reali per programmi di una certa consistenza scritti in Pascal, si sono ritenuti come operatori unici tutte le strutture di controllo presenti nel programma. La curva ottenuta con i dati ricavati seguendo le convenzioni sopra decsritte, contrassegnata con NU1 in figura 1 conferma la miglior stima di N, soprattutto per programmi grossi.

A partire dalla comprensione di questi andamenti è possibile provare a raffinare il modello, che comunque deve essere inteso come mezzo per stimare, con un certo grado di approssimazione, la caratteristiche medie di una classe di algoritmi e non valori puntuali per determinati programmi.

Classi di programmi	Num. di prog.	Valore Medio N	Valore Medio $\hat{N}$	Valore Medio NF	Valore Medio NU1
50 : 100	49	7.06939E+01	1.11017E+02	7.63238E+01	9.49302E+01
100 : 200	61	1.39590E+02	1.76738E+02	1.25569E+02	1.73820E+02
200 : 300	33	2.53000E+02	2.64371E+02	1.92791E+02	2.81949E+02
300 : 400	28	3.41714E+02	3.14735E+02	2.32168E+02	3.59525E+02
400 : 600	45	4.95733E+02	4.17959E+02	3.13122E+02	4.93853E+02
600 : 800	20	6.94250E+02	6.32339E+02	4.86360E+02	7.83752E+02
800 : 1000	13	8.87923E+02	7.04964E+02	5.44337E+02	8.91259E+02
1000 : 1400	11	1.26800E+03	8.36332E+02	6.51447E+02	1.17112E+03
1400 : 2000	14	1.71707E+03	1.08901E+03	8.58566E+02	1.54293E+03
2000 : 3000	12	2.30950E+03	1.20026E+03	9.49546E+02	1.80406E+03

Tabella 4 Valori medi della lunghezza dei programmi, dei dati osservati e delle formule applicate per stimarli

——————— N Curva osservata
- - - - - - - - - - - $\hat{N}$ Curva proposta da Halstead
— . — . — . — . NF Curva di S., C. e D.
— — — — — NU1 Curva di J. e L.

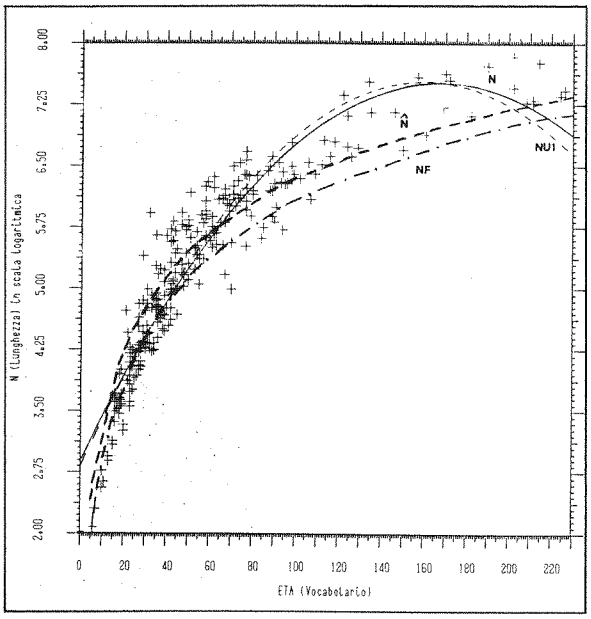


Fig. 1 Andamento della curva ottenuta con i valori osservati, presentati in scala logaritmica, e delle curve proposte quale sua stima

7. BIBLIOGRAFIA

[1] M.H. Halstead, ELEMENTS OF SOFTWARE SCIENCE, Elsevier-North Holland, New York, 1977

[2] R. Boher, HALSTEAD'S CRITERION AND STATISTICAL ALGORITHMS, Proceedings of the Eight Annual Computer Science (Statistics Interface Symposium) Los Angeles, Febraury, 1975

[3] A.W. Bulut, S. Necdet, M.H. Halstead and B.D. Bayer, EXPERIMENTAL VALIDATION OF A STRUCTURAL PROPERTY OF FORTRAN ALGORITHMS, Computer Science Department Technical Report 11, Purdue University, January 1974

[4] J.L. Elshoff, MEASURING COMMERCIAL PL/1 PROGRAMS USING HALSTEAD'S CRITERIA, Sigplan Notices, Vol. 1, No. 5, 1975

[5] J.L. Elshoff, AN INVESTIGATION INTO THE EFFECTS OF THE COUNTING METHOD USED ON SOFTWARE SCIENCE MEASUREMENTS, ACM Sigplan Notices, 13 (2), 1978

[6] D.B. Johnson, A.M. Lister, A NOTE ON THE SOFTWARE SCIENCE LENGHT EQUATION, Software Practice and Experience, Vol. 11, 1981

[7] V.Y. Shen, S.D. Conte, H.E. Dunsmore, SOFTWARE SCIENCE REVISITED: A CRITICAL ANALYSIS OF THE THEORY AND COMPUTER PROGRAM COMPLEXITY, Rome Air Development Center, Griff AFB, Tech. Rep. RADC-TR-78-4, Vol. II, 1978

ELECTRONIC MAIL SYSTEMS AND COMMUNICATIVE BEHAVIOUR

Graziella Tonfoni

Istituto di Glottologia

Università di Bologna

RIASSUNTO

Questo articolo presenta i risultati di un'analisi svolta dell'Autrice negli USA in riferimento specifico alle conseguenze comportamentali e comunicative dell'introduzione di Sistemi di Posta Elettronica nell'ambito delle organizzazioni aziendali.
I fenomeni riscontrati e considerati particolarmente significativi in quanto generalizzabili sono stati interpretati nell'ambito di un modello interpretativo che considera le dinamiche dell'agire comunicativo proprie dell'Ufficio come attività di generazione di campi di attività comunicative di "agenti" comunicatori.

ABSTRACT

The present paper shows the results of a case-study analysis aimed toward checking behavioural and communicative consequences of Electronic Mail Systems on Users within an Office Environments.
It also presents some guidelines for interpreting the phenomea which have been observed and selected as being significant and generalizable.
A positive understanding and use of any technological tool derives from a broader and deeper interpretation of communicative skills within a specific context. Guidelines and proposal have been derived from both observation and modelling of the specific Office Environment.

1. INTRODUCTION

This paper presents the results of a research project lead by the Author aimed toward checking the impact of Electronic Mail Systems within Office Environments. The research has been done within a number of Corporations both in the U.S.A., in the Boston and Combridge area, Massachusetts.

The specific cases, which have been analyzed, have been chosen as being particularly significant and they all share the following kind of features:

1) The working environment is represented by a tipical Office Automation Environment;

2) People working in the Office haven't been previously exposed to the use of any technological tools or Electronic Mail System;

3) The training the Users have been undergoing is purely technical and mainly oriented toward the use of the actual specific System;

4) People working in the Office have been specifically asked to use the System;

5) The System (Electronic Mail) has been introduced without any previous discussion with the actual Users and without checking how felt about introducing it into the Office Everyday's life;

6) Specific Office Environments which have been analyzed present a high density level of personal computers which is, in most cases, one computer for one person;

7) All Electronic Mail Systems which have been analyzed have been introduced and used at least for one year (generally longer);

8) All Electronic Mail Systems which have been analyzed are relatively flexible ones and relatively easy to use after an appropriate training, which all of the Users have been exposed to.

2. THE THEORETICAL BACKGROUND

The explicit goal of any Electronic Mail System is to make communication more effective within an Office Environment. Effectiveness can be thought of as speed in sending and receiving messages, in contacting people working within and outside Office Environments, in organizing time and work in a more satisfactory way both on a personal and on a collective level.
Our view of communication within an Office is based on an interpretation of Office Activities as Network of Committments created by people by the use of language [5].

By following such perspective, communication within the Office can be thought of as action and effectiveness in communication, therefore it involves understanding and sharing the committments everybody is making throught language. Effectiveness in communication also means mutual understanding by Communicators (both Speaker and Hearer) in the specific situation they are being already part of (institution, sharing of experiences, previous agreements) or building up new agreements, (creating new ideas and turning them into projects for possible achievements). Our assumption is that no Electronic Mail System can be separated from the evironment it is both part of and operating on by progressively reshaping and expanding it.
Any Electronic System is an evolving part of an evolving Structure (Office) within another structure (Corporation, Society); it therefore involves sharing of Cultural Background and Beliefs.

Any Office as well as any Structure is not an omogeneous entity, therefore both communication and interaction among individuals are far away to be reduced to a set of preconceived and preplanned behavioral skills and are not likely to be constrained into a limited domain of strategies by the use of an exclusive tool and throught an exclusive code and channel. An "effective Communicator" within an Office Environment is that individual who is able to observe and analyze the communicative context he is presently in, the nature of the communicative domain he is acting on, the goals he is planning to achieve or building up interactively with another or more Communicators by cooperating or negotiating, being therefore part of the dynamic process.

Being the technological tool in this specific case the Electronic Mail System it has to interpreted as a tool for enlarging the domain of possibilities in building up and supporting effective interactions among people in an office or in a broader network.
Interactions among people are different in nature, therefore Communicative Skills have to be different and to adapt to different Contexts depending on different Communicators acting in different domains.
Interactions can be formal and structured, therefore based on common sharing of expectations and ways and tools of communication, like meetings, or informal like occasional conversations, corridor talks, or even gossip (the so called grapevine-communication).

Office Communication is characterized by the coexistence of different kinds of interactions which demand different kinds of skills.

The nature of the very same message varies depending on the way it is being expressed and created by the use of language, either on an oral or a written level. More precisely, within the domain of writing we have to draw a distinction between handwritting a note, or sending messages through the Electronic Mail System; within the verbal domain we also have to distinguish among calling up immediately, leaving a message on the phone, trying to interact personally through conversations, which can be either formal or informal, and more or less structured or spontaneous.

Office Communication is complex; any technological tool which works within an office environment must handle complexity, rather than reduce it by constraining the possible and by imposing a predetermined way of communicating as being generally valid; therefore acquiring new skills means handling different ways of communicating through different tools by recognizing at each time what is good for what and when. It also implies being subject to change whenever it is needed, given the communication interactive and dynamic nature.

What the skilled Comminucator actually does within an Office Environment is action through language by constant perception of the environment through observation and understanding.
Action through communication is a dynamic process; interaction among individuals can be thought of at different levels; it can be on a one versus one or one to more individuals level. The two levels inherrently imply and demand different attitudes, background sharing and linguistic skills.

Shortly, the Skilled Communicator should know how to talk to whom about what, depending on what for. All that is implicitely given in a so called "skilled communicative behaviour" can be made transparent through a certain interpretation of the technological tool, specifically the Electronic Mail System. By using such interpretation, we are viewing the System not as a structure of predefined procedures in order to speed up things which have to get done fast, but as an open structure which helps the User be aware of what he is doing just by deciding to use such system for communicating and not going through other equally possible ways.
If we were here to briefly summarize the framework we are working in these are the points we are referring to:

— Communication in an Office is Action among individuals;

— Action is dynamic and continuously subject to change;

— A Communicator must be skilled on different levels not just on the technical one;

— Any technological communicative tool, which is in an Office, has to be seen as a support tool and not a normative tool which states its own rules;

— Communication effectiveness cannot be determined a priori but it consists on a process and has to be checked at each given time, given the context which is being created through language;

— Effectiveness in Communication cannot be referred to the technical quality of a Technological Tool, it rather has to be referred to the understanding Communicators have of themselves, of the context and of the tool.

The observations which have been in the course of our case study and Office Environment Analysis refer to Behavioural Communicative Consequences after introduction and use of Electronic Mail Systems.

3. DATA ANALYSIS AND OBSERVATIONS

The results which are being presented here as well as the conclusions, are based on a case studyes (a) through observation, (b) from data, (c) by interviewing.
In all such domains, we have been interpreting situations, and kinnds of behaviour (a); observations made about other people behaviour (b); observations made by people themselves (c). This is important to characterize the domain we are actually operating in.

We have been also differentiating between different kinds of evaluations which can be given about the same phenomenon, meaning the role of an Electronic Mail System in the Communication Dynamics within an Office Environment, and precisely:

a) the perception the individual has of himself in relation to the machine, while communicating through it;

b) the perception the individual has of the others while communicating with them through the System.

Point a) involves problems of self confidence in relation to the System performance and technical skill acquisition whereas poit b) involves problems of selfestime and acceptance by building up and keeping a good interactive context through the use of the System.
Referring to point a) and b) we can list here some of the most relevant recurring therefore generalizable facts we have encountered in the course of our analysis.

1) Increased speed of message producing and parallel increased number of message sending;

2) Increased interest into technical features displayed by the system (sophisticated word-processing);

3) Increased selfconfidence in trying new graphical "hacks";

4) Decrease of interest in message reading, given the large amount of those which are sent everyday;

5) Decrease in distinguishing what is relevant and what is not, what to be kept and what has to be deleted;

6) Tendency to sending any kind of message through the system rather than using the phone or talking personally, or even having a secretary send the message;

7) Decrease of style concerns in producing a message, which tends to be short and to be based on jargon and abbreviations; the punctuation has almost desappeared (just question marks and explanation marks persist) as well as capital initial letters;

8) Most of the informal communication gets to be transmitted in a standard code which is common only to a limited group of people;

9) Electronic Mail System Standard Message Format tends to become the officially accepted one. Number of words and lenght of sentences get to be determined by System and tend to influence verbal message passing too (brevity and jargon);

10) Decrease of informal message passing as in corridor conversation; informal communication gets to be incorporated by the system too;

11) Tendency to avoiding long term planning and preference to short term planning where the systems provide an agenda.

Through a pretty wide and detailed set of case-studies, we can generalize and derive the following assertations:

1) Electronic Mail Systems Users, when highly skilled in terms of use of all the technical features exploited by the System, tend to lower their interest level for correct and appropriate linguistic expressions.

It necessary follows that:

2) The more sophisticated the System is the less sophisticated the message style gets to be.

4. REMARKS AND PROPOSAL

In order to account for such observed consequences in the User's communicative behaviour, we'll be referring to the theoretical background we have presented in paragraph 2. Misunderstanding of the nature and meaning of an Electronic Mail System leads into non-committmental behaviour and passive attitudes toward the system. In other words, the system is considered to be the communication source, which determines code-strategies and format-selection, instead of being thought of as tool which can make transparent something which is being already in the user's mind and in the linguistic organization of a message.

A misleading interpretation of the tool leads toward refusal of the complexity which is inherently part of any communication process; complexity and multiple strategies get to be considered as obstacles against so called effective communication.
In other words, a lack of communicative context interpretation and understanding leads to uniformity in communicative strategies and behaviour.
Language and linguistic behaviour result into static and normative code use or attitudes.

No real interaction exists between the user and the system, unless the system is actually conceived as a possible tool among other possible tools for interacting.
An exclusive use of the system for any kind of message to any kind of user indenpendently on the nature of the specific communicative act, on the content and on the goal leads into a collapsing of different communicative styles and strategies.

In order to keep a sharing of common belief which in necessary for any kind of positive interaction between communicators, the system has to be creating new agreement by revising already existing common belief and updating with what is missing.

Writing a message through an electronic mail system is not just typing sentences and identifying the adressee, it rather means for the User to be choosing among different ways of creating message; among different styles, between the oral and the written level, between a formal and an informal level; among differing communicative partners, between keeping the message open or accessible for a larger number of receivers. Any of such choices got specific implications.
None of them is given by the tool itself, it is communicative interaction.

Stressing on the technical aspect of an electronuc mail system as being the most relevant one for successful use, has led (as comes out of our case-analysis) toward repetitive, purely technically skilled behaviour, which leads to executive choice and exclusive use of the system itself without developing any kind of complex and skilled communicative competence.
A possible solution which comes out from our case-analysis can be briefly summarized by the following set of assertions:

1) Never introduce any electronic mail system into an Office Environment without having the users ready and willing to be using it;

2) Never introduce any electronic mail system presenting it as "the solution" to communicative problems;

3) Never encourage the use of any electronic mail system as the exclusive or the best tool for comunicating;

4) Always train the users on a technical level add coaching on communicating with and without electronic mail system support;

5) Always present an electronic mail system as an opportunity and not the only way for communicating more effectively;

6) Always encourage multiple choices and ways of communicating as well as focus on interpreting and understanding which is good for what, when and for whom.

To summarize, the present paper intends to present a mixed approach between a theoretical framework of conceiving technological tools in communication [5] and a methodological framework for introducing such tools into Office Environments and for interpreting what is already happening in "the real world" where such tools have been already introduced.

APPENDIX

The following set of figures shortly summarized our analysis and proposal.

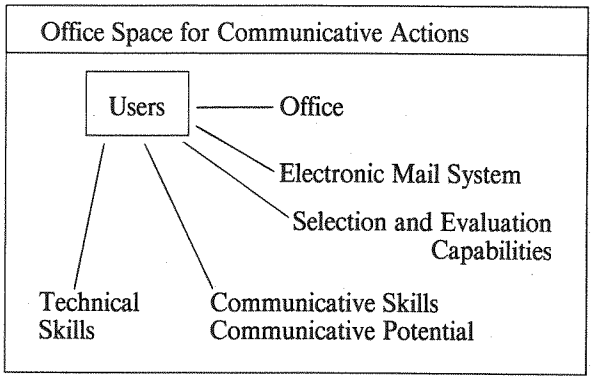


Fig. 1

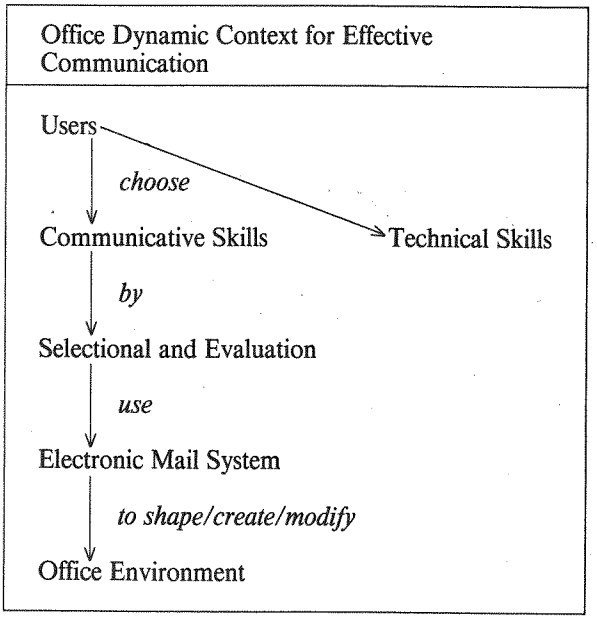


Fig. 2

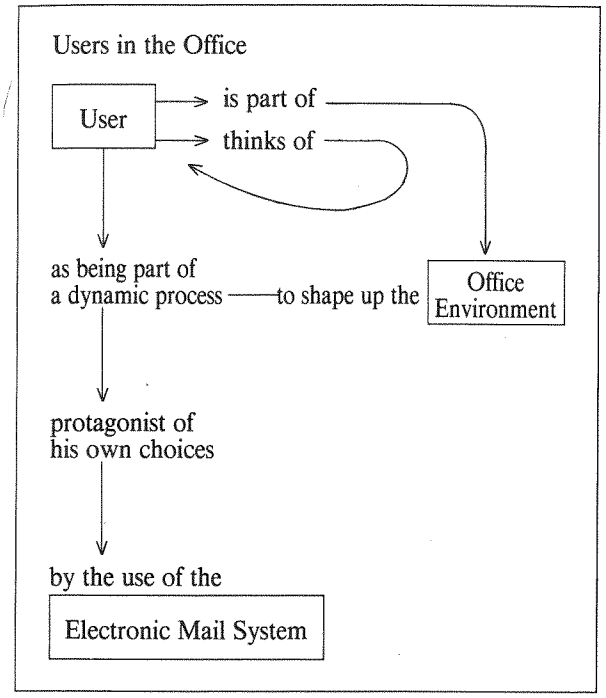


Fig. 3

5. REFERENCES

[1] G. Alessandrini, G. Tonfoni, RETI E PROCESSI: COMUNICAZIONE E FORMAZIONE NELL'AZIENDA, to appear F. Angeli, Milano, 1986

[2] G. Tonfoni, COMUNICAZIONE E INNOVAZIONE TECNOLOGICA, documento Enidata, Milano, 1986

[3] S. Turkle, THE SECOND SELF: COMPUTERS AND THE HUMAN SPIRIT, Simon and Schuster, New York, 1984

[4] T. Winograd, WHEN WILL COMPUTERS UNDERSTAND PEOPLE?, in Psychology Today, 7, 12, pp. 73-79, 1974

[5] T. Winograd, F. Flores, UNDERSTANDING COMPUTERS AND COGNITION, Ablex Publ., Norwood, NJ, 1986

[6] M. Zelany, AUTOPOIESIS, A THEORY OF THE LIVING ORGANIZATION, Elseiver-North-Holland (ed), New York, 1978

UNA COMPONENTE LINGUISTICA PER UN SISTEMA DI QUESTION ANSWERING

M. Rosa Savoldi

Dipartimento di Scienze dell'Informazione

Università di Milano

RIASSUNTO

In questo articolo si presenta un lavoro in cui è stato realizzato un sistema di dialogo uomo-macchina basato su un linguaggio pseudonaturale. Questo linguaggio che è stato chiamato E (da English) è un linguaggio intermedio tra l'inglese e il linguaggio logico Prolog [2], [1]. È stato chiamato pseudonaturale proprio perchè ha certe caratteristiche non "naturali" che lo avvicinano a un linguaggio formale come per esempio l'uso di parentesi per eliminare eventuali ambiguità. Il linguaggio E consente lo sviluppo di un dialogo uomo-macchina durante il quale il sistema acquisisce due tipi di conoscenza: a) definizioni sun un determinato campo di conoscenza; b) definizioni sulla sintassi e sul lessico di E.

ABSTRACT

This paper deals with a man-machine communication system, based on a pseudonatural language. This language, called E (from English), is an intermediate language between English and the logic programming language Prolog [2], [1].It has been called pseudanatural just because it has some not "natural" features so that it seems like a formal language as by example the use of parenthesis to avoid some ambiguites. The E language makes it possible to generate a man-machine dialogue so that the machine learns two different types of know-ledge: a) definitions about a special field of knowledge; b) definitions about sintax and lexicon of E.

1. PROLOG E IL FORMALISMO DELLE DCG

Il linguaggio di programmazione logica Prolog è stato sviluppato nel '71 da Colmerauer ed altri all'Università di Marsiglia ed è stato subito applicato all'elaborazione del linguaggio naturale. L'aspetto dichiarativo viene computato da un programma rispetto a "come" viene computato. Per esempio il programma seguente:

```
append ([],X,X)
append ([A | T],B,[A | S]) : — append (T,B,S).
```

computa la concatenazione di due liste e rappresenta una definizione dichiarativa piuttosto che una procedurale. Tale aspetto dichiarativo consente la scrittura di parsers che risultano trasparenti rispetto al meccanismo di computazione svincolando così il programmatore dal diretto controllo del processo di parsing. Il formalismo più diffuso su cui si basano tali parsers Prolog è quello delle DCG (Definite Clauses Grammers) [4] che sono direttamente traducibili in programmi Prolog.

La forma delle produzioni di una DCG è di tipo context-free dove però:

1) i non-termini qui sono dei termini composti (p.e. sentence (X));
2) la parte destra di una procedura è composta da terminali, non-terminali e chiamate procedura.

Vediamo un esempio:

```
sentence (X) ——> noun__phrase (X), verb__phrase (X).
noun__phrase (X) ——> determiner, noun (X).
verb__phrase (X) ——> verb (X).

determiner ——> [the].
noun (singular) ——> [cat3].
noun (plural) ——> [cats].
verb (singular) ——> [mews].
verb (plural) ——> [mew].
```

Questa grammatica accetta come corrette le frasi "the cat mews" e "the cats mew". Il programma Prolog corrispondente che rappresenta effettivamente un parser che accetta queste due frasi è:

```
sentence (SO,S,X) :— noun__phrase (SO,S1,X), verb
__phrase (S1,S,X).
noun__phrase (SO,S,X) :— determiner (SO,S1), noun
(S1,S,X).
verb__phrase (SO,S,X) :— verb (SO,S,X).

determiner ([the |S],S).
noun ([cat | S],S,singular).
noun ([cats | S],S,plural).
verb ([mews | S],S,singular).
verb ([mew | S],S,plural).
```

È importante notare che anche se una DCG presenta produzioni in forma context-free la sua potenza espressiva è maggiore in quanto può, tramite l'aggiunta di argomenti ai simboli non terminali, trattare problemi contestuali, trasformazioni. ecc. Infatti nell'esempio precedente si risolve tramite l'argomento X la dipendenza del verbo dal soggetto cosicché se il soggetto è singolare (plurale) anche il verbo deve essere singolare (plurale).

Le DCG consentono la scrittura di parsers sia per l'analisi sintattica di frasi in linguaggio naturale sia per la generazione di frasi in linguaggio naturale. Tramite l'aggiunta di argomenti ai simboli non-terminali si possono anche costruire strutture complesse associate alla frase da analizzare come parse trees, rappresentazioni logiche, ecc.

2. IL LINGUAGGIO E

Il linguaggio E è nato come linguaggio intermedio tra l'inglese e il Prolog. Quindi può essere visto sotto due aspetti:

1) come sottoinsieme limitato ma significativo dell'inglese; limitato perchè comprende solo frasi dichiarative (affermative/negative) con qualche tipo di subordinata (condizionali, relative) e frasi interrogative (semplici, composte); significativo perchè è sufficiente per esprimere definizioni su un determinato campo di conoscenza;

2) come un "Prolog naturale" nel senso che avvicina alla programmazione in Prolog.

Le frasi di E sono solo di tipo singolare (III persona). Ogni frase dichiarativa può essere affermativa o negativa. Anche per le frasi interrogative esiste la forma interrogativo-negativa.

Il tempo presente indicativo viene usato per esprimere proprietà non eventi. Così la frase:

"john goes to school"

vuole dire che "john" ha la proprietà "goes th school" mentre se si vuole esprimere l'evento bisogna dire

"john is going to school"

dove si fa uso del tempo presente progressivo.

Questa scelta permette di stabilire univocamente il significato di una frase. Lo stesso discorso vale anche per la frase inglese ambigua

"a bee flies"

dove l'ambiguità in E viene risolta così:

1) "a bee is flying" (se si vuole esprimere un evento);
2) "every bee flies" (se si vuole esprimere una proprietà).

È importante notare che l'articolo "a" è tramutato in "every" in quanto nella frase "a bee flies" usata per esprimere una proprietà ha valore universale e non esistenziale. Infatti in E l'articolo "a" (come pure "an", "the") deve essre usato solamente quando ha valore esistenziale come per esempio nelle frasi

"john owns a car"
"john is a student".

Ad ogni frase dichiarativa affermativa corrisponde una negativa, un'interrogativa, un'interrogativa-negativa:

F.D.AFF. "john goes to school"
F.D.NEG. "john does not go to school"
F.I. "does john go to school"
F.I.NEG. "does not john go to school?".

Oltre a frasi di tipo dichiarativo E comprende frasi subordinate come le condizionali e la relative. Le condizionali sono introdotte dalla congiunzione "if"; le relative sono introdotte dai pronomi relativi "who" e "which" preceduti da quantificatore di tipo universale come "anyone", "everyone", "anything", "everything", "every (NOUN)". Consideriamo la frase seguente:

X is the aunt of Y if
X is the sister of Z and
(Z is the mother of Y or
Z is the father of Y).

L'uso di parentesi rotonde e la presenza di parole come X, Y, Z sono i due fattori principali per cui il linguaggio E è detto pseudonaturale.
Infatti son proprio queste due caratteristiche che lo avvicinano al linguaggio logico Prolog: le parentesi rotonde servono per eliminare ambiguità di associazione tra le congiunzioni "and" e "or" mentre le parole X, Y, Z rappresentano le variabili nel senso logico.

Le frasi relative di E possono essere comunque espresse anche sotto forma di frasi condizionali.

Esempi:

a) REL: mary like anyone who likes music.
CON: mary likes X if X likes music.

b) REL: everyone who is a student goes to school.
CON: X goes to school if X is a student.

c) REL: every student who studies logic likes logic.
CON: X likes logic if X is a student and X studies logic.

La prima frase di b) in E può essere espressa anche così:

"every student goes to school".

È evidente che le frasi relative corrispondono esattamente a frasi del linguaggio naturale mentre le corrispondenti frasi condizionali sono più vicine a Prolog.

3. DA E A PROLOG

Le frasi dichiarative di E che non contengono quantificatori vengono rappresentate internamente sotto forma di fatti Prolog. Le frasi dichiarative che contengono quantificatori e le frasi che contengono delle subordinate (relative e condizionali) vengono rappresentate internamente sotto forma di regole Prolog. Le frasi interrogativa di E vengono rappresentate internamente sotto forma di domande Prolog.

La corrispondenza tra E e Prolog è analoga alla corrispondenza tra frasi in linguaggio naturale e calcolo dei predicati [3]. Il Prolog è un linguaggio del calcolo dei predicati che ammette però solamente clausole di Horn, cioè clausole che hanno al massimo un letterale positivo.

Le clausole di horn si presentano in una delle tre forme seguenti:

1) A
2) A <— B1 / \ .. / \ Bn
3) <— B1 / \ .. / \ Bn

dove 1), 2), 3) rappresentano rispettivamente fatti, regole, domande Prolog a A e Bi (i > O) sono letterali positivi. Una regola Prolog rappresenta un'implicazione logica (B —> A) dove però il conseguente precede l'antecedente (la freccia è scritta nel senso contrario). Il conseguente viene detto "testa" ("head") della regola mentre l'antecedente (che può essere un singolo letterale o una congiunzione di letterali) si dice "coda" ("tail"). Se sono presenti delle variabili nella testa ed eventualmente nella coda, queste sono intese quantificate universalmente mentre se delle variabili si presentano solo nella coda sono intese allora quantificate esistenzialmente.

La regola generale di traduzione da una frase dichiarativa/interrogativa di E in un fatto/domanda Prolog è che il verbo diventa il predicato della formula mentre soggetto e complemento oggetto (o altri complementi) diventano gli argomenti del predicato.

E: john sing.
PROLOG: sings (john).
E: john likes mary.
PROLOG: likes (john, mary).
E: john goes to school.
PROLOG: goes__to (john, school).

Valgono gli stessi criteri per le corrispondenti frasi interrogative:

E: does john sing?
PROLOG: ?— sings (john).
E: does john like mary?
PROLOG: ?— likes (john, mary).
E: does john go to school?
: ?— goes__to (john, school).
dove l'operatore '?—' del Prolog sta ad indicare che ciò che segue è una domanda.

Per le frasi dichiarative col verbo "to be" usato come copula abbiamo la seguente traduzione:

E: john is a student.
PROLOG: is__a (student (john)).
E: john is the father of mary.
PROLOG: is__the (father__of (john, mary)).

Le frasi che contengono condizionali e relative diventano regole Prolog.

Per esempio:

E: john likes X if X is a singer.
PROLOG: likes (john, X) :— is__a (singer (X)).
E: X is the aunt of Y if X is the sister of Z and (Z is the father of Y or Z is the mother of Y).
PROLOG: is__the (aunt__of (X, Y)) :— is__the (sister__of (X, Z)), (is__the (father__of (Z, Y)); is__the (mother__of (Z, Y))).
E: every student goes to school.
PROLOG: goes__to (X, school) :— is__a (student (X)).
E: mary likes anyone who studies music.
PROLOG: likes (mary, X) :— studies (X, music).

Una nota particolare meritano le frasi negative. Infatti una frase negativa (cioè contenente l'avverbio di negazione "not") viene tradotta in fatto Prolog dove il predicato è "non" (che rappresenta l'avverbio "not") e il singolo argomento del predicato "non" è la corrispondente frase affermativa.

Così abbiamo:

E: john does not sing.
PROLOG: non (sings (john)).

Questo predicato vuole simulare il "not" logico ma non coincide con esso perchè non è un connettivo ma un semplice predicato.

4. UN SISTEMA DI Q/A

Tramite il linguaggio E è possibile creare un vero e proprio ambiente di question/answering che risulta molto simile al sistema di question/answering del Prolog.

Una domanda Prolog viene "computata" in questo modo: il sistema cerca di dimostrare la validità (cioè se è vera) della formula associata alla domanda in base all'insieme di clausole correnti che costituiscono il database del sistema, cioè cerca di dimostrare che tale formula è una conseguenza logica dell'insieme di clausole che rappresentano gli assiomi del sistema.

Questa procedura risulta in una dimostrazione per refutazione cioè la formula associata alla domanda viene negata e aggiunta all'insieme di clausole di cui si deve dimostrare che è una conseguenza logica. Se viene dimostrata l'inconsistenza di questo nuovo insieme di clausole allora si è dimostrata la validità della formula iniziale. Se un insieme di clausole è inconsistente vuol dire che si è ricavato attraverso l'applicazione di regole di inferenza la clausola vuota o nil che rappresenta l'assurdo. Il Prolog utilizza come unica regola di inferenza il principio di risoluzione [5] che risulta completa per clausole di Horn. Se nella domanda compaiono delle variabili il valore di tali variabili viene computato attraverso un algoritmo di unificazione (o sostituzione) in modo tale che per questi valori la formula associata alla domanda rappresenti effettivamente una conseguenza logica degli assiomi di partenza.

Supponiamo di avere un database composto dalle clausole seguenti:

```
athenian (socrates).
athenian (plato).
greek (X) :— athenian (X).
```

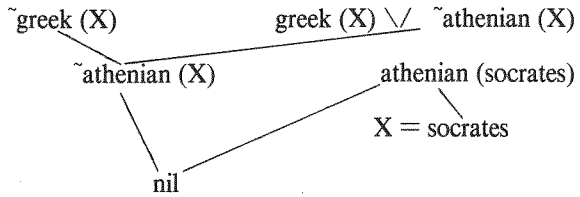
Alla domanda:

```
?— greek (X).
```

il sistema risponderà

```
X = socrates
more (y/n)?
```

avendo infatti unificate durante la dimostrazione per refutazione la variabile X con il termine "socrates". Il grafo seguente rappresenta tale dimostrazione.



Il sistema chiede se si vogliono altre risposte; in caso affermativo attua un processo di "backtracking".

Per spiegare che cosa significa "backtracking" bisogna introdurre un secondo modo di vedere le regole Prolog cioè attraverso un'interpretazione di tipo procedurale. Sotto tale interpretazione la testa di una regola è vista come l'intestazione di una procedura mentre la coda è vista come il corpo di una procedura. Quando deve soddisfare una domanda cerca una clausola con la stessa testa dopo di che la coda della clausola diventa a sua volta una congiunzione di domande. Se a un certo punto non riesce a soddisfare una di queste domande allora viene attuato un meccanismo di backtracking col quale il sistema risoddisfa la più recente domanda per cui esistono più alternative (cioè più clausole con la stessa testa).
Nell'esempio precedente se viene attuato il backtracking dopo la risposta X = socrates il sistema risoddisfa la domanda "athenian (X)" attraverso il fatto "athenian (plato)" unificando allora la variabile X con il termine "plato".

Se il sistema è riuscito a soddisfare una certa domanda allora risponde "yes", altrimenti risponde "no" intendendo dire con questa risposta che non è riuscito a dimostrare la validità della formula associata alla domanda e perciò ha dimostrato la validità della sua negata. Si dice infatti che il Prolog ammette una negazione per insuccesso ("negation by failure") cioè tutto ciò che non viene dimostrato è falso.

Il sistema di question/answering realizzato tramite il lingiaggio E si comporta in modo diverso; se una certa domanda D non è stata soddisfatta tenta di soddisfare la domanda "non (D)" (dove "non" è il predicato introdotto precedentemente per rappresentare le frasi negative di E); se fallisce anche questo allora il sistema risponde "I don't know". Quindi la risposta "no" di questo sistema per una domanda D significa che è stata soddisfatta la domanda "non (D)" e non semplicemente come in Prolog che la domanda D non è stata soddisfatta e quindi è falsa.

Supponiamo di avere inserito nel sistema le frasi seguenti:

```
*> john is a student.
*> mary is not a teacher.
*> every student does not work.
```

Queste frasi rappresentano il database del sistema. Ora si può avere il dialogo seguente:

```
*> is john a student?
   yes, it is true that john is a student.
*> is mary a teacher?.
   no, it is not true that mary is a teacher.
*> is mary a student?
   I don't know.
*> who is a student?
   john is a student.
   other answer?.
*> y
   I don't know.
*> who does not work?
   john does not work.
   other answer?
*> n
*> who is a student.
   other answer?
*> n
   ....
   ....
```

Le frasi di E del database sono rappresentate internamente in questo modo:

```
is_a (student (john)).
non (is_a (teacher (mary))).
non (works (X)) :— is_a (student (X)).
```

Con la rappresentazione "is_a (student (john))" è possibile fare una domanda del tipo "who is john?" mentre con la rappresentazione "student (john)" questa domanda non sarebbe possibile.

La frase "other answer?" simula il "more (y/n)?" del Prolog e a risposta affermativa il sistema attua un processo di backtracking per cercare eventuali altre risposte.

5. E: UN SISTEMA ESTENSIBILE

Il sistema è stato interamente implementato in Prolog. Come mostrato nella fig. 1 il sistema è composto da

a) un parser per le frasi dichiarative + le subordinate e un parser per le frasi interrogative che hanno in comune oltre ad alcune regole sintattiche il dizionario di E (tali parsers sono basati sul formalismo delle DCG);

b) un traduttore da frasi di E in clausole Prolog che incrementa il database del sistema nel caso di una frase dichiarativa mentre le frasi interrogative vengono passate a un interprete Prolog-like che le "computa".

c) un sottosistema composto da un parsers (+ vocabolario) e un traduttore che consente di estendere automaticamente il vocabolario e il parsers di E sotto opportune limitazioni.

L'estensione automatica avviene tramite un dialogo con l'utente. Ogni volta che si inserisce una frase il sistema ne fa l'analisi sintattica e la passa sotto forma di 'parse tree' al traduttore. Quando una frase non è riconosciuta dal parser possono esservi due situazioni:

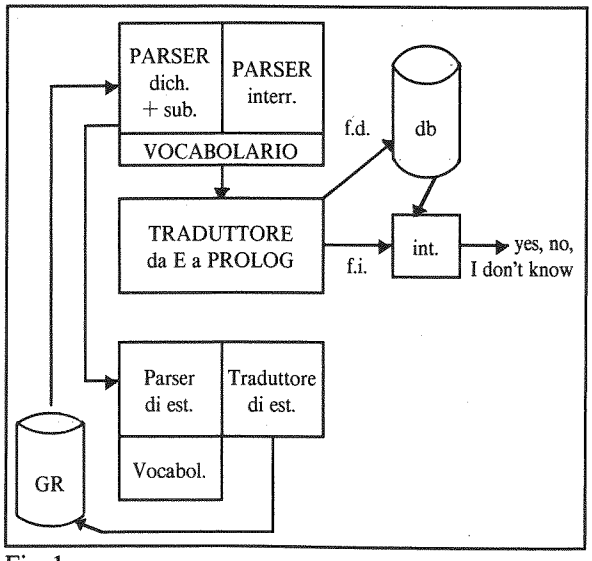


Fig. 1

1) il sistema non conosce alcune parole della frase
2) il sistema non conosce una delle regole sintattiche che generano la frase.

Il sistema attuale prevede la possibilità di estensione per i seguenti tipi di frasi:

1) nominali (come "john", "a student")
2) verbali (come "goes to school", "likes mary", "sings")
3) predicative (come "is a student", "is good").

con la possibilità di aggiungere i seguenti tipi di parole:

1) nomi (propri e comuni)
2) aggettivi (qualificativi)
3) avverbi
4) servili
5) preposizioni
6) verbi.

L'introduzione di preposizioni è permessa solo nelle frasi verbali e predicative ma non nelle frasi nominali perchè si vuole evitare di avere una grammatica ambigua. Infatti la frase in linguaggio naturale

"john listens to the noise in the street"

è ambigua in quanto la preposizione "in" può legarsi sia alla frase nominale che la precede ("the noise") sia la verbo della frase ("listens").
Le frasi trattate hanno al massimo un complemento indiretto.

Supponiamo di introdurre nel sistema la frase seguente:

"john lives in italy".

Il sistema non riconosce questa frase e si avvia allora il dialogo seguente:

```
what is lives?
*> a verb.

INFINITE FORM:
*> live.

is live a regular verb?
*> y

PRESENT: lives.
PAST_PARTICIPLE: lived.
GERUND: living.

is live a transitive verb
*> n

what is in?
*> a preposition.

what is italy?
*> a noun.
```

I cannot parse the phrase:
john lives in italy.
which rule do you want to extend?
1) noun__phrase.
2) verbal__phrase.
3) predicative__phrase.
*> verbal__phrase.
The rules defining a verbal__phrase are the following:
.....
..... {vengono visualizzate le regole correnti che definiscono una verbal__phrase}
Give me the new rule about a verbal__phrase
*> a verbal phrase is composed by an intransitive verb and preposition and a noun phrase.
Ok: the phrase "john lives in italy" is correct.
.....
.....

I nuovi vocaboli e la nuova regola sintattica diventano ora correnti cosicchè possono essere usati nella continuazione del dialogo e vengono direttamente riconosciuti. In questo modo il parser e il vocabolario di E vengono ingrementati interattivamente.

È importante notare che il traduttore da E a Prolog è composto da regole generali di traduzione che valgono anche per quelle frasi che non vengono riconosciute dal parser iniziale di E.

7. CONCLUSIONE

Il sistema di dialogo realizzato consente di avvicinarsi alla programmazione in Prolog attraverso un linguaggio pseudonaturale dichiarativo e interrogativo; viene messo in luce l'aspetto dichiarativo del Prolog e la stretta relazione tra Prolog e linguaggio naturale o pseudonaturale.

8. BIBLIOGRAFIA

[1] W.F. Clocksin, C.S. Mellish, PROGRAMMING IN PROLOG, Springer Verlag, 1984

[2] H. Gallaire, A STUDY OF PROLOG, in Bierman, Guiho (Ed.), "Computer program synthesis methodologies", D. Reidel Pu. Co., 1981

[3] N.J. Nilsson, PRINCIPLES OF ARTIFICIAL INTELLIGENCE, Springer Verlag, 1982

[4] F. Pereira, DEFINITE CLAUSES GRAMMARS FOR LANGUAGE ANALYSIS, Artificial Intelligence 13, 1983

[5] J.A. Robinson, A MACHINE-ORIENTED LOGIC BASED ON THE RESOLUTION PRINCIPLE, J. ACM 12, 1982

PROTOTIPIZZAZIONE ED EXPERT SYSTEM SHELL: UN ESEMPIO D'USO DI Xi

Silvano Marioni

Banca Solari & Blum S.A.

Lugano

RIASSUNTO

L'articolo illustra l'impiego di un expert system shell come strumento per costruire e sperimentare un modello che sarà poi sviluppato con un linguaggio di programmazione tradizionale. Viene proposto un caso pratico nel settore della tariffazione delle commissioni di borsa sviluppato con l'impiego dell'expert system shell Xi.

ABSTRACT

This paper presents the applications of an expert system shell for building and testing a model before developing and programming with a programming language. The case study is showing an application using the expert system shell Xi in the area of stock exchange regulations.

1. INTRODUZIONE

Le continue modifiche, tipiche dei processi di manutenzione del software, richiedono una costante evoluzione dei programmi applicativi e spesso l'introduzione di ulteriori cambiamenti può portare alla necessità di revisione di una intera applicazione.
Negli ultimi anni si è investito molto nello studio e progettazione di linguaggi che permettono di ottenere il risultato di un problema descrivendo le sue regole senza dover necessariamente indicare il loro ordine di applicazione (fig. 1). Uno dei risultati più importanti a cui porta l'adozione di linguaggi di questo tipo è una maggiore facilità sia nei processi di sviluppo che di manutenzione del software.

Il processo di sviluppo di un programma comporta in una prima fase la definizione di un modello che comprenda la

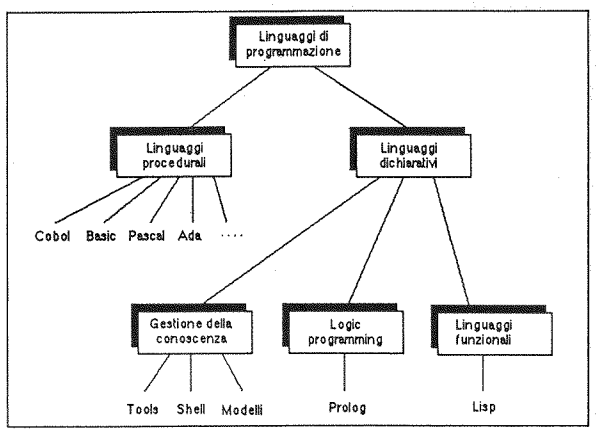


Fig. 1 Linguaggi di programmazione procedurali e dichiarativi

conoscenza del problema.
Dopo la formalizzazione del modello è possibile affrontare la fase che porta alla creazione dell'algoritmo. Solo a questo punto si è in grado di iniziare a scrivere le istruzioni in codice.
Ma che cosa succede se, dopo aver ideato e programmato l'algoritmo, scopriamo che la nostra conoscenza del problema non era completa?
Non è facile aggiungere i pezzi di conoscenza mancante a quella che abbiamo già congelata nell'algoritmo perchè ci si scontra con le difficoltà che un linguaggio di programmazione procedurale, cioè sequenziale, ha nell'elaborare una sitazione complessa di tipo dichiarativo. Questi inconvenienti si incontrano con sempre maggior frequenza perchè la carenza di comunicazione tra utente e programmatore è aumentata con il crescere di complessità

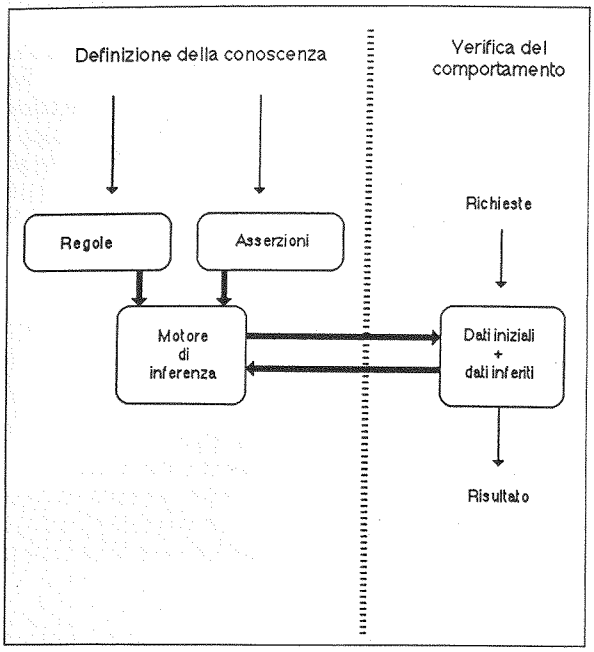


Fig. 2 Definizione della conoscenza con un Expert System Shell

dei sistemi informativi. L'utente che desidera un programma per risolvere un suo problema ha spesso una visione limitata e parziale perchè non ha più il possesso diretto dei dati e non è in grado di descrivere un modello adeguato del suo problema. Il programmatore d'altro canto non riesce a comprendere il problema se non dalla descrizione data dall'utente. Se a tutto questo aggiungiamo le ambiguità del linguaggio quando si affrontano problemi di una certa complessità, non è difficile immaginare che gli errori nella comunicazione della conoscenza possono incidere in modo determinante nei tempi e nella qualità del lavoro di programmazione.

Per ovviare a questi inconvenienti la soluzione più efficace è quella di provare in anticipo il problema con strumenti che permettano di verificare il modello in un modo completo e che rendano semplici gli eventuali processi di revisione della conoscenza.

Tra gli strumenti che oggi permettono di applicare la metodologia della prototipizzazione, i linguaggi per la gestione della conoscenza, e tipicamente gli expert system shell, non richiedono la definizione di un algoritmo a priori, ma permettono una definizione estremamente modulare della conoscenza come regole separate una dall'altra (fig. 2).

Questa caratteristica permette di iniziare il processo di costruzione e di verifica aggiungendo nuove regole quando servono e rendendo possibile la definizione dell'algoritmo in modo incrementale, senza avere ancora definito il modello.

Un expert system shell diventa uno strumento di indagine del modello che sarà sviluppato in seguito in un linguaggio di programmazione tradizionale.

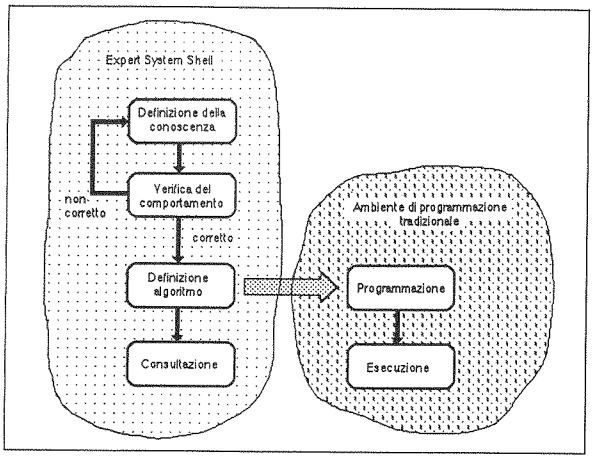


Fig. 3 Regole definite nel sistema esperto e nel programma tradizionale

Si può porre a questo punto l'obiezione che gran parte degli algoritmi di un programma riguardano le parti più tecniche, quali la gestione delle videate o l'rganizzazione e i metodi di ricerca dei dati.
L'expert system shell non si pone come un metodo per costruire in precedenza tutti gli algortimi di un programma, ma solo quelli di una discreta complessità e che devono essere verificati con la collaborazione di un esperto.

Alla fine del lavoro di progettazione dell'algoritmo uno degli effetti collaterali da non trascurare è quello di avere costruito un vero e proprio sistema esperto nel settore di applicazione dell'algoritmo. La conoscenza del problema fornita dall'esperto e operante nel programma tradizionale, è presente anche in un nuovo modo nel sistema esperto a cui si può fare riferimento per modificare e sperimentare l'algoritmo o per utilizzarla come fonte di documentazione (fig. 3).

2. L'EXPERT SYSTEM SHELL Xi

Xi è un ambiente di programmazione della conoscenza funzionante su un personal computer che permette la costruzione e la consultazione di un sistema esperto. È la versione commerciale e migliorata di Props, un sistema sviluppato presso l'Imperial Cancer Research Fund Lab., uno dei maggiori centri di ricerca inglesi. Xi è composto da uno shell e da una serie di strumenti per facilitare la memorizzazione e la gestione della conoscenza. Sviluppato e commercializzato attualmente dalla ditta Expertech, è scritto in Prolog-2 e funzione su un P.C. con almeno 380 Kb di memoria. Il linguaggio di programmazione di Xi permette la composizione di regole e di asserzioni memorizzate in forma esplicita nel calcolatore con una sintassi molto vicina alla lingua inglese. È possibile una gestione dell'input e output che comprende la generazione automatica di menu per l'entrata dati, la gestione dei report di output sia su video che su printer, l'accesso a file su disco. Il motore di inferenza permette di adottare sia la tecnica del forward chaining che quella del backward in modo da permettere un ottimizzazione nella consultazione della base di conoscenza.

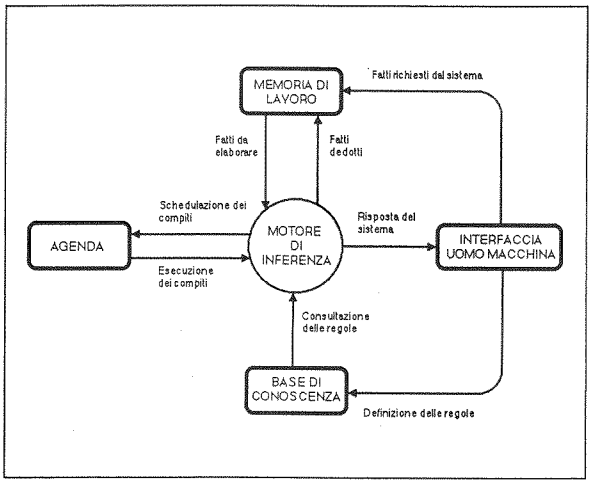


Fig. 4 Architettura dell'Expert System Shell Xi

Progettato volutamente come strumento utilizzabile da persone non esperte di informatica può essere usato in due modi a seconda della necessità di impiego. Il modo "menu", si presta per il lavoro di consultazione della base della conoscenza e permette un utilizzo estremamente semplice del sistema esperto utilizzando i menu come strumento di dialogo per fornire tutti i dati richiesti. Il modo "dialogo" è un modo più sofisticato di fare consultazione ma soprattutto è il modo per definire e introdurre la conoscenza e costruire il sistema esperto.

Xi permette di costruire sistemi esperti utilizzando il metodo delle regole di produzione. L'architettura del sistema esperto è composta da una parte statica, la conoscenza memorizzata, e una dinamica, le situazioni che si vengono a creare nel contesto di una consultazione (fig. 4).

La parte statica è formata da un insieme di conoscenze su un particolare dominio che è espresso sotto forma di regole e fatti viene chiamato comunumente "base di conoscenza".

La parte dinamica è formata da una memoria di lavoro dove vengono poste tutte le conclusioni derivate dalle regole per essere riutilizzate, in fasi successive, fino al raggiungimento dell'obiettivo: la gestione di questa parte della dinamica è operata dal motore inferenziale.

Xi dispone inoltre di una ulteriore parte chiamata "agenda", che sovraintende al controllo delle linee di ragionamento.
Vale forse la pena di ribadire il modo in cui le regole e i fatti della base di conoscenza sono utilizzati in un sistema esperto, per poter dare una giusta collocazione a Xi. Esistono in effetti due strategie che corrispondono a due modi di ragionamento. Nel primo caso si possono fornire a priori tutti i fatti di cui si è a conoscenza. Se alcuni di questi fatti fanno rendere vera una regola questa aggiunge alla memoria di lavoro la sua conclusione e questo può far ricominciare un nuovo processo di deduzione fino a quando non è più possibile dedurre nuovi fatti. Questo modo di procedere chiamato forward chaining è guidato dai fatti ed è tipico del ragionamento deduttivo.

Nel secondo caso il meccanismo funziona all'inverso. A partire da un fatto fornito come obiettivo, il sistema tenta di soddisfare le condizioni che portano alla sua verifica. Può darsi il caso che una delle condizioni sia una conseguenza di un'altra regola e questo porta il motore di inferenza ad iniziare un nuovo processo all'indietro alla ricerca delle nuove condizioni da verificare. Quando tutte le premesse sono state soddisfatte il fatto definito come obiettivo è vero. Questo modo di procedere chiamato backward chianing è guidato dagli obiettivi ed è tipico del ragionamento per ipotesi.

Xi permette di utilizzare entrambi i metodi di ragionamento e dispone di comandi che permettono di schedulare e di eseguire le linee di ragionamento secondo le strategie che l'esperto ritiene più opportune.
Questa possibilità di funzionamento misto, che utilizza l'agenda come strumento di controllo, permette di scomporre il problema in aree definite e soprattutto di assegnare la priorità di risoluzione delle varie aree a dipendenza del contesto.

Xi si presta per essere ustato facilmente da un programmatore che descrive le regole e le asserzioni per risolvere il problema senza tener conto dei problemi di controllo ma dedicandosi solo al loro contenuto logico.
Contemporaneamente anche l'utente come esperto del problema è in grado di verificare tutti i possibili comportamenti, isolando i malfunzionamenti, e permettendo al programmatore di moficare in modo rapido il modello del problema.

3. UN ESEMPIO DI USO DI Xi

La Banca Solari & Blum ha utilizzato l'expert system shell Xi come strumento di prototipizzazione in occasione dell'introduzione della nuova Convenzione di borsa svizzera alla fine del 1985.

Il modello della nuova convenzione è chiaramente definito in un documento del Vereinigung Schweizerischer Effektenbörsen che specifica i modi e i tipi di operazioni di borsa e fissa i tassi da applicare all'operazione. Per le operazioni concluse in Svizzera sono stabiliti dei tassi in funzione del tipo del titolo e del valore di borsa della transazione (Art. 9.1 - 9.6). A questo si deve aggiungere per le operazioni concluse all'estero e fino al controvalore di Fr. 250.000 un nuovo tipo di logica che fa dipendere i tassi dalla nazione (Art. 10.1 - 10.2). Quando il controvalore supera Fr. 250.000 si riprende la regola delle operazioni concluse in Svizzera con l'aggiunta delle commissioni estere effettive (Art. 11). Se l'operazione conclusa all'estero è al netto di commissione estera si applicano le regole per le operazioni concluse in Svizzera (Art. 12). La convenzione dà inoltre indicazioni precise per classificare i tipi di titolo in funzione della nazionalità o della divisa o di altre caratteristiche del titolo.

Questo non vuol essere il riassunto della convenzione di

borsa svizzera ma è solo una descrizione di quella parte di logica che verrà approfondita nel resto dell'articolo. Dal confronto con un estratto della convenzione di borsa (fig. 5) può essere interessante notare come, con problemi di una certa complessità, possono sorgere delle difficoltà di comprensione della conoscenza, già formulando il testo con parole diverse.

Coscienti di questi tipi di difficoltà si è iniziato il lavoro di prototipizzazione con l'intento di scrivere un sistema esperto che raccogliesse tutte le regole della convenzione di borsa svizzera.

Dopo il lavoro di costruzione si è iniziato un processo di verifica e di revisione del sistema esperto con la collaborazione attiva degli esperti di borsa. Al termine di questo lavoro è stato possibile trascrivere in un tempo molto breve tutta la logica nel relativo programma di COBOL. Anche il tempo dedicato alla fase di test del programma è diminuito notevolmente poichè gran parte del test della logica era già stato effettuato durante la costruzione del sistema esperto.
La tecnica adottata è stata quella di iniziare da un prototipo che comprenda le direttive principali per poi aggiungere, in un secondo tempo, i dettagli ai diversi livelli gerarchici. È importante sottolineare che questa tecnica di tipo strutturato è utile solo come metodo di analisi del problema ma non nella scrittura delle regole perchè una delle caratteristiche principali di un expert system shell è quella di poter operare con basi di regole poco stabili.

Nel sistema esperto sulle convenzioni di borsa l'obiettivo

Estratto dalla Convenzione sulle commissioni di borsa dell'Associazione svizzera delle Borse valori (valida dal 31 dicembre 1985).

Art. 9
Per le operazioni concluse in Svizzera vigono le seguenti tariffe scalari secondo valori lordi per transazione:

	Operazioni concluse in Svizzera
9.1 Azioni svizzere e titoli analoghi	
— sino a fr. 50.000.-	0,8%
— per i successivi fr. 50.000.-	0,7%
— per i successivi fr. 50.000.-	0,6%
— per i successivi fr. 150.000.-	0,5%
— per i successivi fr. 300.000.-	0,4%
— per i successivi fr. 400.000.-	0,3%
— per i successivi fr. 1 mio	0,2%
— per l'importo eccedente fr. 2 mio.	negoziabile
9.2 Azioni estere e titoli analoghi	
— sino a fr. 50.000.-	1,0%
— per i successivi fr. 50.000.-	0,9%
— per i successivi fr. 50.000.-	0,7%
— per i successivi fr. 150.000.-	0,5%
— per i successivi fr. 300.000.-	0,4%
— per i successivi fr. 400.000.-	0,3%
— per i successivi fr. 1 mio.	0,2%
— per l'importo eccedente fr. 2 mio.	negoziabile
9.3 Obbligazioni in franchi svizzeri	
— sino a fr. 50.000.-	0,6%
— per i successivi fr. 50.000.-	0,4%
— per i successivi fr. 200.000.-	0,3%
— per i successivi fr. 700.000.-	0,2%
— per i successivi fr. 1 mio.	0,1%
— per l'importo eccedente fr. 2 mio.	negoziabile
9.4 Obbligazioni in monete estere	
— sino a fr. 50.000.-	0,6%
— per i successivi fr. 50.000.-	0,5%
— per i successivi fr. 50.000.-	0,4%
— per i successivi fr. 150.000.-	0,3%
— per i successivi fr. 700.000.-	0,2%
— da fr. 1 mio. (limite d'esenzione)	totalmente negoziabile
9.5 «Notes» e «Schuldscheine»	
— sino a fr. 50.000.-	0,6%
— per i successivi fr. 50.000.-	0,5%
— per i successivi fr. 50.000.-	0,4%
— per i successivi fr. 100.000.-	0,3%
— da fr. 250.000.- (limite d'esenzione)	totalmente negoziabile
9.6 Parti di fondi d'investimento	
— sino a fr. 500.000.-	0,3%
— per i successivi fr. 500.000.-	0,2%
— per i successivi fr. 1 mio.	0,1%
— per l'importo eccedente fr. 2 mio.	negoziabile

Art. 10
Per transazioni «al lordo» concluse all'esteno sino al controvalore di fr. 250.000.- valgono le seguenti tariffe globali (inclusive delle commissioni estere) scalari secondo i singoli valori lordi per operazione:

10.1 Azioni	sino a fr. 50.000.-	per i successivi fr. 50.000.-	per i successivi fr. 150.000.-
Stati Uniti	2,5%	1,5%	1,0%
Giappone	2,0%	1,5%	1,3%
Rep. fed. di Germania	1,4%	1,3%	1,2%
Inghilterra	2,7%	1,7%	1,3%
Paesi Bassi	1,8%	1,7%	1,6%
Canada	2,7%	1,7%	1,3%
Australia	2,0%	1,5%	1,3%
Europa, altri	1,6%	1,5%	1,4%
Oltremare, altri	2,5%	1,5%	1,0%

Tariffe globali per transazioni «al lordo» concluse all'estero sino a fr. 250.000.-

10.2 Obbligazioni	sino a fr. 50.000.-	per i successivi fr. 200.000.-
Tutti i paesi	1,5%	0,8%

Art. 11
Alle transazioni «al lordo» concluse all'estero per importi superiori al controvalore di fr. 250.000.- sono applicabili le aliquote minime di commissione previste dall'art. 9 per la corrispondente categoria di titoli, maggiorate della commissione estera.

Transazioni «al lordo» concluse all'estero per importi superiori a fr. 250.000.-

Art. 13
Per le operazioni previste dagli art. 9 e 10 vigono i seguenti importi minimi:

Importi minimi

a) operazioni concluse in Svizzera:
— azioni, obbligazioni — minimo fr. 30.-
— parti di fondi d'investimento — minimo fr. 10.-

b) operazioni concluse all'estero:
— netto e lordo — minimo fr. 80.-

c) operazioni su obbligazioni in monete estere, «Notes» e «Schuldscheine» — minimo fr. 80.-

Per tutte le operazioni con un valore lordo inferiore a fr. 250.-: 10%, massimo fr.10.-

Fig. 5 Estratto della Convenzione di borsa svizzera

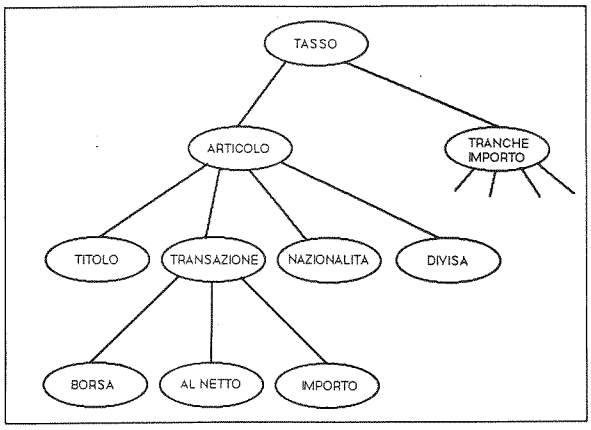


Fig. 6 Struttura di una parte della base di conoscenza

finale è la determinazione dei tassi di commissione da applicare nelle diverse operazioni previste dalla convenzione.

Il sistema è previsto per fornire come risultato della consultazione, la suddivisione della quantità totale e i relativi tassi da appliacare a ogni tranche.

Procedendo in modo gerarchico (fig. 6) sono identificate come direttive principali la scelta del tipo di articolo per stabilire i tassi da applicare (identificatore *articolo*) e della tranche di importo minimo per determinare le sudduvisioni della quantità totale a cui applicare i tassi (identificatore *tranche importo*).

La regola seguente significa che alla richiesta iniziale del tasso deve essere valutato con un processo di backward chaining, prima l'obietivo *articolo* e poi l'obiettivo *tranche importo.*

if query tasso
then check articolo
  and check tranche importo

Non indaghiamo nella direzione di *tranche importo* poiché si tratta di un'area troppo vasta e poco adatta per essere analizzata a questo livello.
Le regole necessarie per definire l'identificatore *articolo* sono un buon esempio della facilità di impiego di Xi come strumento di formalizzazione della conoscenza nel settore delle leggi e dei regolamenti.

La figura 7 mostra le relazioni che il motore di inferenza stabilisce nel grafico corrispondente all'esempio presentato qui di seguito.

if titolo **is** azioni
  and transazione **is** svizzera
  and nazionalità **is** svizzera
then articolo **is** 9-1

if titolo **is** azioni
  and transazione **is** svizzera
  and nazionalità **is** estera
then articolo **is** 9-2

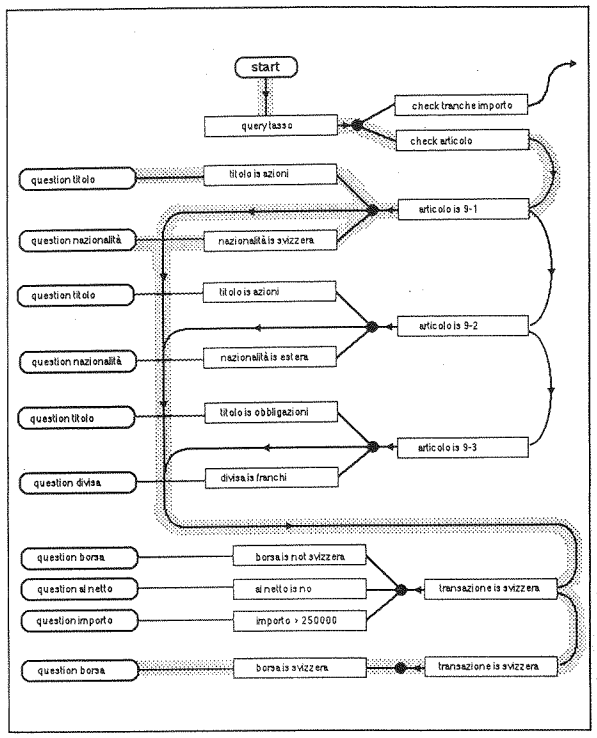


Fig. 7 Schema di funzionamento di una parte della base di conoscenza

if titolo **is** obbligazioni
  and transazione **is** svizzera
  and divisa **is** franchi
then articolo **is** 9-3

In ognuna delle regole precedenti secondo il processo del forward chaining la conclusione si verifica se tutte le condizioni sono vere.
Gli identificatori *titolo, nazionalità, divisa* sono riconosciuti come dati forniti dall'esterno e il sistema li imposta ponendo le domande relative all'utente. Le istruzioni che permettono di assegnare uno dei possibili valori all'identificatore sono elencate qui di seguito.

question titolo
Quale è il tipo di titolo
one of
azioni,
obbligazioni,
notes,
parti

question nazionalità
Quale è la nazionalità del titolo
one of
svizzera,
estera

question divisa
Quale è la divisa in cui è rimborsato il titolo
one of
franchi,
divisa estera

L'identificatore *transazione* per essere verificato necessita una linea di ragionamento differente. Il motore di inferenza trova che gli unici punti dove può essere valutato sono delle conseguenze di altre regole e quindi inizia un processo di backward chaining per soddisfarle. Le regole interessate sono definite qui di seguito

if borsa **is** svizzera
then transazione **is** svizzera

if borsa **is not** svizzera
  and al netto **is** no
  and importo <= 250000
then transazione **is** estera

if borsa **is not** svizzera
  and al netto **is** no
  and importo > 250000
then transazione **is** estera

if borsa **is not** svizzera
  and al netto **is** si
then transazione **is** svizzera

Anche in questo caso gli identificatori *importo, al netto,* e *borsa* sono riconosciuti come dati forniti dall'esterno e quindi vengono impostati ponendo delle domande all'utente.

question importo
Imposta l'importo in franchi svizzeri
range 0 **to** 8.000.000

question al netto
L'operazione è al netto della commissione estera
one of
si,
no

question borsa
In quale borsa è trattato il titolo
one of
svizzera,
usa,
giappone,
inghilterra,
. . . .
. . . .

Una volta definito l'identificatore *articolo* il motore di inferenza procede con dei processi analoghi alla definizione di *tranche importo* e contemporaneamente alla visualizzazione del risultato.
Le istruzioni appena viste possono servire per dare un'idea a livello più tecnico del linguaggio usato nell'expert system shell Xi.
Questo è solo un campione dei circa 200 elementi, quali regole, fatti, e domande, che compongono la base di conoscenza e che sono state ottenute come risultato del lavoro comune di impostazione e di revisione svolto dal programmatore e dell'esperto di borsa.

4. UN'ESPERIENZA PRATICA

Un punto delicato nel processo di formalizzazione dell'algoritmo è dato dalla necessità di interfacciare le nuove logiche con il sistema informativo esistente. È naturale affrontare il problema con una forma mentale che porta a vedere le nuove modifiche in funzione dei modelli di conoscenza presenti nell'azienda. Le nostre aspettative ci fanno assumere come acquisito ogni concetto che ha un alto grado di somiglianza con i concetti che ci sono familiari. Spesso si tralascia di approfondire il concetto nella lettura delle nuove specifiche e si rischia di non rendersi conto delle differenze che, anche se appaiono in modo sfumato, possono essere fondamentali. Un esempio di questa possibilità di errore può servire come chiarimento; quando nella convenzione di borsa si parla di obbligazioni in franchi svizzeri si è portati a pensare alla divisa in cui è trattata in borsa l'obbligazione. Una lettura più attenta della convenzione specifica che si deve considerare la divisa in cui il titolo sarà rimborsato alla scadenza. Il fatto che nella maggior parte dei casi la divisa di trattamento e la divisa di rimborso sono le stesse facilmente può trarre in inganno e portare a considerare solo la divisa legata al titolo.

Questo tipo di errore "di disattenzione" può essere evitato utilizzando in prima stesura nel sistema esperto gli stessi testi della convenzione di borsa e sicuramente, riprendendo la logica in fase di creazione dell'algoritmo, sarà più facile prestare attenzione alle differenze.

Questa che può sembrare un'osservazione marginale è invece un punto che merita di essere approfondito nell'ottica di utilizzo di un expert system shell. Il lavoro di analisi che tradizionalmente viene fatto prima di iniziare a scrivere il programma è di tipo statico e richiede il passaggio alla fase di programmazione per ottenere una verifica.
L'expert system shell offre delle possibilità in più. La logica può essere ricopiata direttamente dalle specifiche alla base di conoscenza, ma quello che più conta può essere anche verificata immediatamente in modo dinamico facendo funzionare il prototipo di sistema esperto fino ad ottenere il comportamento desiderato.

L'estrema semplicità con cui le disposizioni possono essere riprese dal testo e scritte nel linguaggio di un expert system shell è una delle grosse novità rispetto allo stile di programmazione tradizionale.
La conoscenza è espressa in forma esplicita. Se trascuriamo il fatto che alcune parole chiave sono espresse in lingua inglese possiamo notare che la leggibilità delle regole è molto vicina a quella del testo originale.
Questa caratteristica permette di utilizzare le regole per dare spiegazioni sulla linea di ragionamento che il sistema esperto sta seguendo. Possiamo chiedere spiegazioni sul "come" una certa conclusione è stata raggiunta e sul "perchè" una certa domanda viene posta. La risposta sarà data con la visualizzazione della regola o della serie di regole che in quel preciso contesto sono prese in considerazione dal motore di inferenza.
Le regole sono indipendenti una dall'altra, vengono impostate senza preoccuparsi del loro ordine di esecuzione e

possono essere corrette senza nessun pericolo di alterare la sequenza di esecuzione.
Non è necessario lo scrivere le regole in modo strutturato. Nella base di conoscenza le regole sono raggruppate secondo alcuni criteri logici ma questo viene fatto solo per rendere più chiara la lettura.

Alcuni expert system shell (Xi è uno di questi) hanno la possibilità di effettuare un controllo di coerenza logica delle regole nella base di conoscenza. Diventa possibile fornire le regole in modo frammentario e utilizzare il sistema come uno strumento di aiuto per stabilire le corrette relazioni che nei problemi di una certa complessità logica spesso restano nascoste.

5. CONCLUSIONE

I problemi per i quali è possibile definire un algoritmo di risoluzione non sono in genere considerati di competenza dei sistemi esperti.
Un expert system shell, come nuova tecnologia di programmazione, è preferito nei casi in cui la conoscenza è meno definita e dove la soluzione è ottenuta applicando delle regole di tipo euristico, perchè la struttura logica non è stata ancora chiarita e formalizzaœta. Sembra quindi una contraddizione risolvere un caso di programmazione tradizionale, dove tutto può essere definito con precisione, con uno strumento studiato soprattutto per affrontare problemi per i quali non sia noto con esattezza come arrivare ad una soluzione. In effetti la costruzione di un algoritmo passa attraverso momenti in cui la visione del problema non è sempre molto chiara.

In queste situazioni non esattamente formalizzate esiste un grosso spazio per l'applicazione di un expert system shell, che può essere di aiuto per passare dalla descrizione di un meccanismo logico all'algoritmo che lo riproduce, attraverso un'evoluzione di modelli successivi.

6. BIBLIOGRAFIA

[1] S. Bandini, I SISTEMI ESPERTI, "Note di software", n. 23/24, Giugno 1984

[2] C. Donalisio, PROGRAMMAZIONE LOGICA E PROLOG, "Informatica Oggi", Aprile 1986

[3] C. Chiopris, P. Krauth, A. Markus, LINGUAGGIO PER (SISTEMI) ESPERTI, "Informatica Oggi", Maggio 1986

[4] P. Szeredi, PROLOG THE NEW GENERATION LANGUAGE FOR APPLIED A.I., Atti del convegno "Intelligenza Artificiale: strumenti e applicazioni in ambiente Prolog", Padova, 1984

[5] J. Fox, A SHORT ACCOUNT OF KNOWLEDGE ENGINEERING, Imperial Cancer Research Fund Lab., London, 1985

[6] P. Jackson, INTRODUCTION TO EXPERT SYSTEM, Addison-Wesley Publishing Co, Reading, Massachusetts, 1986

[7] D.A. Waterman, A GUIDE TO EXPERT SYSTEM, Addison-Wesley Publishing Co, Reading, Massachusetts, 1986

[8] R. Schank, L. Hunter, THE QUEST TO UNDERSTAND THINKING, "Byte", Aprile 1985

[9] A. d'Agapeyeff, THE EXERCISING AND SHARING OF DECISION KNOW-HOW IN BUSINESS, Expertech Ltd., Slough, 1985

[10] Xi REFERENCE MANUAL, Expertch Ltd., Slough, 1985

[11] Courtage-Konvention, Vereinigung Schweizerischer Effektenbörsen, Zürich, 1985

AVVENIMENTI

THE 2nd ADVANCED COURSE ON PETRI NETS

Fiorella De Cindio, Lucia Pomello

Dipartimento di Scienze dell'Informazione

Università di Milano

ABSTRACT

In this report tha main areas considered in the 2nd Advanced Course on Petri Nets are identified and for each of them both the state of the art reached on occasion of the course, and some hints about the directions of research and development which result more promising are indicated and discussed.

RIASSUNTO

In questo report vengono identificati i principali argomenti trattati al 2° "Advanced Course on Petri Nets" e per ciascuno di essi viene indicato e discusso il livello di consolidamento raggiunto in occasione del corso e vengono delineate le direzioni di ricerca e sviluppo che risultano più promettenti.

The 2nd Advandec Course on Petri Nets was held in Bad Honnef (Germany) from September 8 to 19, 1986, organized by GMD under the auspices of AFCET, AICA, BCS, EATCS and GI.

33 lectures were given by 26 Lecturers to an audience consisting of 130 partecipants coming from industrial and academic environments, half already pursuing research or application in the field of Petri Nets, the other half involved in the wider area of distributed computing and thus intersted in being introduced into the Petri Nets world. The nice efficiency of the Course Directors and of all the staff allowed this complex machine to run in an intense and quiet atmosphere suitable for facilitating learning, fruitful discussion, informal talks and the necessary relaxation.

Given the number of lectures and lecturers and the breadth of the communications content, we cannot mention all of them. Therefore, we will simply identify six areas indicate, for each one, both the point of assessment reached on occasion of the course, and some hints about the directions of research and development which now result more promising. The main success and usefulness of the Course consists in having attained of this twofold result.

Furthermore two important pebblications are now or in the near future available. On the one hand the Course lecutures will be published by Springer Verlag: they give everyone aiming to work with Petri Nets a well established and common basis, for overcoming a number of difficulties and problems left open by the 1st Advanced Course (Hamburg, September 1979). On the other hand the collection, in the GMD Report n° 212, of the list of the papers corcerning nets theory and applications published up to now is a further very useful support which to a great extent solves the problem of the spread literature.

Let us now come to the six areas. Please, in the following do not consider the space in terms of lines of text proportional to the space given to the topic in the Course: it depends rather on our ability to synthesize! Sometimes we will use quotations from the papers.

1) After nearly 25 years from the Petri's thesis, taking into account that the technological advances makes it "no longer ... extravagant to talk of 64K or more processors being concurrently active at a single task", "it seems that tha notion of Concurrency should attract some attention,

since it is the only basis notion which distinguishes sequential processing from multiprocessing" (here the quotations are from the Petri's opening lecture).

The 2nd Advanced Course shows the progress resulting from the work done in these years by Petri himself and by his school, in particular on the properties of discreteness, density and continuity. This makes now this matter, which is, for the understanding of concurrency, basic as well as once it was harsh, in the range of a wider audience. At the same time, Petri himself warns that the theory is still not complete, at least since "the assumptions on the dimension and topology of the Space and Statespace are missing entirely". These are, therefore, "topics for future research".

2) A serious problem both for new incomers in the field of Petri Nets and for people already engaged in it was the number of classes of nets, presented in the Hamburg Course or little by little proposed by different authors. A major merit of this Course was that of cleaning oup a bit in this sense in different ways:

- first of all, by introducing Condition/Event (C/E) Systems in terms of Elementary Net Systems, the simpler model satisfying the principles which concern the two dual notions of states and changes-of-state and characterize the contribution of Net Theory to the development of a theory of distributed systems;
- secondly, by claiming that Place/Transition (P/T) Nets - "the most common and the most extensively studied class nets in the 1970s" - are no more the central model, mainly because they have ambigous semantics;
- thirdly, in regard to classes of high-level nets, by outlining which are the relationships between Predicate/Transition Nets and Coloured Nets (for both of them being now available a rigorous definition) taking into account the differences in the linear-algebra techniques for the invariant calculus.

In short, the Course proposed Elementary Net Systems and C/E Systems for fundament research, and high-level nets (Pr/T and Coloured Nets) for pratical purposes. In this frame, instead of considering Place/Transition Nets as generalized C/E Systems, they should be seen as a special case of the Predicate/Transition Nets obtained, through suitable projection operators, by ignoring the individuality of the tokens. This new point of view requires that the structural proporties of net systems, and the definition of particular classes of nets, often given in terms of P/T Nets, are set in the new context.
This is for instance the case of important subclasses of P/T Nets, such as Free-Choice Nets, for which the wide range of results and properties have been collected from the spread literature and systematically arranged on occasion of the Course in a very fruitful way.
This is much more true and necessary in the case of some extensions, relevant from the applicative point of view, such as Fifo Nets and the highly controversial Timed/ Stochastic Nets.

3) A third class of lectures concerned the formal tools for studying the behavior of net systems, traditionally performed by means of the so-called Petri Net languages (i.e., by considering the possible sequences of actions) and by means of the notion of process (i.e., by means of Partial Ordered Sets).
In regard to the former, even in this case the Course was a very fruitful occasion for putting together and arranging in a systematic way the main results obtained in these years.
In regard to the latter, attention is now focused on the different approaches for analyzing non-sequential behaviours. Besides Petri's approach based on processes, Mazurkiewicz's Trace Theory and Winskel's Event Structures were presented together with the initial results about their comparison.

4) Comparison with other models was carried on by considering Lauer's and Shield's COSY, Milner's CCS and Hoare's TCSP. This topic was of great interest to the audience, intent on discussing and comparing different and 'popolar' approaches to distributed systems modeling, both from a theoretical point of view (on themes such as concurrency parallelism, non-deterministic simulation, on the notions of observer and experiment) and from an application-oriented perspective (in which equivalence notions are considered as a tool for verifying the consistency of an implementation with respect to a previous specification).

5) The interest in nets in terms of their application for real system modeling was confirmed by the number of lectures covering fields such as: office automation, software engineering, data bases, computer systems, protocols, production systems, computational processes. Tha potentiality already prooved asks for more tools supporting real applications.
One kind of request is for more conceptual tools, allowing, in particular, the possibility of formal manipulations of net models preserving differen properties: liveness, safeness, deadlock-freeness, behavioural equivalences.
But the major request is for computer-based tools. A request which has already a number of answers.

6) Thanks to a major organizational effort, an extensive exhibit of net tools was open throughout the two weeks of the Course, attracting great interest. From general purpose text&graphic editors to packages for the analisys of properties of classes of nets, from prototypes to final products for different classes of hardware, the selection was vast.
But the state-of-the-art in this area does not consist only in a number of packages. It consists also in a complete charaterization of the main requirements a net-oriented computer-based tool should satisfy. This gives everyone parameters for evaluating both existing and future tools.
The crucial importance of tools development was pointed out by Petri himself in his closing lecture. "The tools are closest to my heart" - he said. Not only as a support for the applications, but as a "prerequisite for Net Theory" as well.

Besides tools, he pointed out some aspects which require more attention: namely, duality, meshes, information-flow graphs. Finally he advised of the need of a precise answer, in terms of net theory, to questions concerning time: more precisely he gave some hints for local clocks modeling and asked for realistic (in the sense of "respecting the limitations of implementors") axioms for timed/-stochastics nets.

The way in which Petri has caught in the whole of the lectures and brought to everyone's attentions the points not sufficently considered or neglected and the open problems shows, once more, his ability for proposing stimula and contributions to the development of informatics. An uncommon ability which makes Petri an outstanding figure of the field.

This uncommon figure was, at mid-course, heartily honored, on occasion of his 60th birthday. In the charming setting of Schloss (Castle) Birlonghoven, inside the GMD, Petri was warmly and enthusiastically acknowledged for his work by representatives of Institutions such as IFIP, GI, EATCS and GMD, by his closest colleagues and by more then one hundred people coming from four Continents (only Africa was not represented), all whose activity has been influnced by his ideas.

IL NOTIZIARIO SUI CD-ROM

a cura di A. Borgia (+°), *G. Dalto* (°), *G. Ferrari* (+°), *C. Pagani* (+°), *F. Vagliasindi* (+°)

• La **Library Corporation,** editrice dal 1974 di DataBase bibliografici, offre sia hardware che software basati su tecnologia CD-ROM. Il sistema chiamato ***"Bibliofile"*** contiene l'intero Database MARC dell'English language Library of Congress. Chiavi di ricerca sono il numero LC, International Standard Book e/o numero seriale, titolo e autore.

• L'inglese **Silver Platter Information Service** offre un servizio "completo" per la produzione, distribuzione e mantenimento di CD-ROM. Questo include preparazione degli indici (inverted files), masterizzazione, replica, imballaggio e distribuzione. È possibile l'acquisto in leasing dello hardware e software necessario.

• Nel Novembre 1985 la **Reference Technology** ha annunciato il ***CLASIX Multidrive Director™*** che aumenta il numero di lettori CD-ROM che possono essere collegati ad un personal computer. Il sistema può connettere fino ad otto lettori CLASIX DataDrives Series 500 ad un singolo PC IBM, o PC/XT o PC/AT o compatibili. Il software di lettura è lo STAF/FILE prodotto sempre dalla stessa ditta. In omaggio viene dato il *CLASIX Software Library Data Plate,* un CD-ROM contenente circa 8800 programmi per il PC IBM.

• La **Battelle** ha prodotto ***Micro Basic,*** una versione per personal e micro computers del più celebre Basis, che permette di gestire 200000 pagine di testo scritto su CD-ROM. Il software gira in ambiente MS DOS e UNIX. La ricerca avviene per parole chiave, frasi, prefissi, suffissi e correlando parole chiave dal contesto.

• La **TMS** ha prodotto un nuovo sistema operativo il che permette ***LASERDOS agli utenti di leggere un disco laser usando gli stessi comandi e procedure dei dischi magnetici. Il sistema distribuito dalla*** **Langton Electronic Publishing Systems** può essere completato da software e servizi per la preparazione di CD-ROM.

• La **Toshiba** ha annunciato un drive per Cd-ROM con un tempo medio di accesso di soli 0.28 secondi, che afferma essere il più veloce al mondo.

• **È stato raggiunto un accordo fra *STET, Philips e Dupont* per la realizzazione della prima fabbrica di CD-ROM in Italia.**

• I partecipanti alla **Wester Library Network** intendono avvalersi dei vantaggi dei servizi di reti senza essere sottomessi al rialzo dei costi delle telecomunicazioni. A questo scopo lo staff della WLN sta cercando un gruppo di prodotti integrati basati sul CD-ROM.

• ***Lasersearch,*** sistema di identificazione e acquisizione di libri prodotto dalla **Ingram** funziona con un compact disk drive connesso ad un PC-IBM. *Lasersearch* incorpora il database ***Library Corporation's ANY-BOOk.***

• **Verbatin** prevede il lancio sul mercato nel 1987 di un ***disco ottico erasable*** con capacità di circa 100 Mbyte e con velocità di trasferimento dati variabile da 1 a 5 Mbits/sec.

• La **SONY Corp.** e la **Knowledge Set. Corp.** hanno annunciato una joint venture rivolta al mercato del data-publishing. Le due società si propongono di convertire database esistenti in formato funzionante su CD-ROM. È prevista anche la produzione di hardware e software per CD-ROM.

• La **Optical Storage International,** una società formata dalla **Control Data** e dalla **N.Y. Philips** per produrre dischi ottici scrivibili, aggiungerà alla sua produzione i CD-ROM.

• **SONY** e **Philips** prevedono entro il primo quadrimestre del 1987 di immettere sul mercato il ***CD-I (Compact Disk Interactive).*** È stato comunque chiarito che CD-I e CD-ROM non si troveranno in competizione bensì saranno il complemento l'uno dell'altro. Infatti il CD-I si rivolgerà al mercato del ***home entertainment.***

• **Engineering Information** e **Digital Equipment Corp.** hanno avviato uno sforzo comune per poter offrire un sottoinsieme del database ***COMPENDEX*** (Computerised Engineering Index) su CD-ROM.

• La **Access Innovations Inc.** e la **Silver Platter Information Inc.** hanno siglato un accordo per lo sviluppo di un prodotto CD-ROM per il ***National Information Center for Educational Media*** **(NICEM),** database correntemente disponibile con servizi on-line.

(°) *Dipartimento di Scienze dell'Informazione*
(+) *Media Group S.r.l.*

RECENSIONI

Pierluigi Fiorani Gallotta, INTRODUZIONE ALL'ARCHITETTURA DEI SISTEMI DI ELABORAZIONE, Clued Editore, Milano, 1984

Lo studio dell'architettura dei sistemi di elaborazione, cioè dell'interfaccia tra hardware e software, è un passo importantissimo per la comprensione degli aspetti fondamentali di un sistema di elaborazione delle informazioni.
Il testo, raccolta delle lezioni del corso di Sistemi II presso il Dipartimento di Scienze dell'Informazione dell'Università degli Studi di Milano, presenta l'argomento mediante due esempi significativi: l'architettura del PDP 11/70 della Digital, e dello Z8000 della Zilog. Esso parte dalla visione che ha un utente finale (programmatore in linguaggi ad alto livello) che usa la macchina a un alto livello di astrazione usando le librerie fornite con il linguaggio, per arrivare alla visione che ha un utente sofisticato (programmatore assembler) che ha una visione completa delle strutture hardware e software della macchina.
Vengono poi presentate le principali operazioni di CPU, il loro costo, il meccanismo degli interrupt, i vari modi di indirizzamento e le operazioni di input/output come sono utilizzati nelle architetture correnti nei mini e nei micro con una certa enfasi sulle interfaccie software che costituiscono il nucleo del sistema operativo e i driver per l'input/output.

Dario Lucarella, DOCUMENTAZIONE AUTOMATICA, Clued Editore, Milano, 1983

La produzione automatizzata di testi mediante word processor e altri strumenti, ha assunto con l'uso sempre maggiore dei personal computer, una diffusione di massa. Ma l'editoria elettronica non si ferma all'uso di un word processor: necessita strumenti sofisticati per l'impostazione della pagina, l'inserimento di formule matematiche, di disegni e figure, per avere un metodo uniforme di produzione di testi.
Dario Lucarella ha lavorato in questo ambito dal 1974 e dal 1982 è stato nominato professore a contratto per il corso di Documentazione Automatica presso la facoltà di Scienze dell'Informazione dell'Università degli Studi di Milano ed è l'autore di questo libro interamente dedicato a questo tema.
Il testo costituisce una raccolta delle lezioni del corso di cui sopra, ed è centrato sul testo come struttura informativa con le sue possibili rappresentazioni, con un orientamento verso i problemi implementativi di Sistemi Integrati per il trattamento (Text Editing, Linguaggi di Markup, TEX e Metafont), l'archiviazione e la distribuzione di documenti (Document Delivery), per chi affronti da punto di vista sistemistico applicativo.
Il testo consente un primo contatto col mondo dell'editoria elettronica che sta ora avendo una forte diffusione grazie alle stampanti laser, ed è pertanto consigliabile a chi voglia conoscere, almeno a grandi linee, la struttura dei vari sistemi di editing esistenti.

a cura di E. Ciapessoni

Honeywell
Honeywell Information Systems Italia
Via Laboratori Olivetti - Pregnana Milanese

UNIVERSITÀ
DEGLI STUDI DI MILANO
DIPARTIMENTO DI SCIENZE
DELL'INFORMAZIONE
Via G. Colombo, 49
20133 Milano - Tel. 2772234

Honeywell NOTE SW.34 M011001 01
Honeywell Information Systems Italia

CIMINO MICHELE

HISI-DIREZIONE GENERALE MARKETING ITALIA

VIA PIRELLI,32
20124MILANO

Ufficio Documentazione Gestione Mailing List
Via del Parlamento, 33 - **20098 BORGOLOMBARDO (MI)**
Trattasi di trasporto di cancelleria-stampati-moduli-documenti interni-mobili e macchine per ufficio ESONERATI DALL'OBBLIGO DELLE BOLLE DI ACCOMPAGNAMENTO D.P.R. 6/10/78 n. 627 art. 4 n. 8 e Circolare Ministeriale n. 72 del 23/12/78 paragrafo 10